The Dissertation Committee for Dimitrios Papailiopoulos
certifies that this is the approved version of the following dissertation:

Distributed Large-scale Data Storage and Processing

Committee:

Alexandros Georgios Dimakis, Supervisor

Constantine Caramanis

Pradeep Ravikumar

Sujay Sanghavi

Sriram Vishwanath

Distributed Large-scale Data Storage and Processing

by

Dimitrios Papailiopoulos, B.E; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2014

This thesis is dedicated to the four most memorable and loving people I was lucky enough to have in my life, my Mother Lilly, my late Grandparents – Vasiliki and Dimitris, and my love Issis. This thesis would not have been made possible without your struggle, love, and support. Thus, it goes wholeheartedly to all four of you.

“ Άξιον εστί το χέρι που επιστρέφει
από φόνο φριχτόν και τώρα ξέρει
ποιός αλήθεια ο κόσμος που υπερέχει
ποιό το “νυν” και ποιό το “αιέν” του κόσμου

...

Νυν η ταπείνωση των Θεών
Νυν η σποδός του Ανθρώπου

Νυν Νυν το μηδέν

και ΑΙΕΝ Ο ΚΟΣΜΟΣ Ο ΜΙΚΡΟΣ, Ο ΜΕΓΑΣ!”

- Οδυσσέας Ελύτης

Acknowledgments

The path to obtaining a doctoral degree can be long and strenuous. In my case it did not feel like either. It was a beautiful and smooth ride, which, if I had a second chance, I would do it all over. In this path many loved ones, friends, and colleagues have helped me along the way. Let me take some time to thank them all.

I would like to start off by acknowledging the support of my advisor, Prof. Alex Dimakis, who has been the best advisor a student could ever hope for. His ingenuity, patience, energy, curiosity, breadth of knowledge, and generosity can rarely be found in a single individual. Under his guidance I enjoyed a rare combination of immeasurable academic care and an absolute academic freedom. Intellectually, I have tremendously benefited from our long meetings, most of which started unscheduled with me casually dropping by his office (he always has his door open to everyone). We would start off with figuring out if we had any "pending items", then I would go on with enthusiastically exclaiming "I think I can prove [insert long-standing problem here]" (with little actual success most of the time). A simple question would then derail the conversation for over an hour to some obscure combinatorial argument in graph theory, with Alex wanting to understand the intuition using toy examples on balls and bins, and me trying to convince him that everything boils down to computing the eigenvectors of some matrix. Many times our discussions would spin off to topics completely irrelevant to research, such as the fundamentals of economy explained through the lens of the "banana salesman and masseur on a deserted island" toy example. Alex has had a tremendous intellectual influence on me, and I can only wish that I inherited some of his crisp academic vision and a finer taste for problems. He was key in making this ride one of the most rich and enjoyable in my life, and for that I thank him. I can only hope to be at least a small fraction as good of an advisor for my future academic children. Alex, I will forever be indebted to you (quite literally at this point).

My academic journey started much earlier, in Greece. I took my first academic steps at the Technical University of Crete (TUC), located in the beautiful city of Chania. The seven years I spent there, first as an undergrad, and then as

a Masters student, have shaped a large part of my character and intellect. TUC offers a rare combination of an engaging academic environment with some of the best views of the Sea of Crete (Kritiko Pelagos). Many acknowledgments go to our professors back there, who persevered in spite of the many difficulties and obstructions in the Greek educational system. They provided us with solid academic background and supported us on our future steps, with great love and care. A great thanks goes to Prof. Nikos Sidiropoulos. His course, Signals and Systems, is the course that shaped my path towards the mathematical foundation of engineering. I wish to also thank Prof. Athanasios Liavas for teaching us the fundamentals of wireless communications, and for being patient with my many interrupting questions. I wish to finally thank Prof. Dimitris Rovas, for his exciting lectures during a graduate course at TUC, that made linear algebra one of the topics I hold dearest to my heart.

I would like to also thank Prof. George Karystinos, my undergraduate and Masters advisor. He is responsible for a great deal of my academic upbringing, as he is the one that directed me towards the art of basic research. I was lucky to have collaborated with him and I owe a great deal of my research toolbox to his ingenious ideas. I still remember the long meetings next to his whiteboard trying to prove that certain channel matrices were constant rank. I want to thank him for convincing me that a Ph.D. in the US was the right path, and for all the invaluable lessons.

Many thanks go to my good old friends from the undergrad, the fun years, Stavros, Constantinos, Nikos, Manolis, and Giorgos. We grew up together, and shared some of the most memorable moments in my life. I would also like to thank my Los Angeles friends that I made during the first years of my Ph.D. at the University of Southern California. Specifically, my old roommate Bill and my friend Stelios that made the first rough years fun. I am immensely indebted to my good Greek-Austinite friends that I met (and reconnected with) when I moved from USC to UT Austin. Niko and Yanni, thank you so much for your friendship, love, and hospitality. I wish that I will be as lucky in my new moves, as I was when I moved to Austin. Without a doubt Austin would not be as fun, if we did not have Constantine's home gatherings. Constantine, thank you for your hospitality and immense generosity. Finally, I have to mention that big part of the Austin experience was being the proud percussionist of the Greek Rembetiko band "Magkes piaste tin Congress Bridge," with all aforementioned Austinites (faculty included).

During my academic years, I was extremely lucky to have collaborated with amazingly smart and talented people. In chronological order of collaboration, many thanks go to Babak Hassibi, Asteris Megasthenis, Viveck Cadambe, Changho Suh, Georgina Abou-Elkheir, Jianqiang Luo, Cheng Huang, Jin Li, Karthik Shanmugam, Guiseppe Caire, Itzhak Tamo, Stavros Korokythakis, Maheswaran Sathiamoorthy, Ramkumar Vadali, Scott Chen, Dhruba Borthakur, Yannis Mitliagkas, Constantine Caramanis, Ankit Singh Rawat, Sriram Vishwanath, Tasos Kyrillidis, Christos Boutsidis.

The greatest of acknowledgments goes to my Mother who taught me to be curious about nature and helped shape my imagination and engineering skills with good old LEGOs. Her struggle was long and tough, but in the end we made it. I want to thank my Father for his support, particularly during the TUC years. I would like to express my deep gratitude for and love towards my late grandparents, that supported me with infinite love and deeply believed in me. All my loved ones deserve more than a simple acknowledgement. I thank you all who played a sweet part in my life and shaped me to be the person that I am. Your names are not mentioned here, but you know who you are.

Last but not least, I want to thank my partner Issis. I find it amazing how wholeheartedly you believe in my dreams and aspirations, and how much you love and support me. Thank you for being part of this path, for being loving, patient with me reading papers most of the time, and amazing in every possible aspect. Your love, splendor, and intellect never cease to amaze me.

Distributed Large-scale Data Storage and Processing

Publication No. _____

Dimitrios Papailiopoulos, Ph.D.
The University of Texas at Austin, 2014

Supervisor: Alexandros Georgios Dimakis

This thesis makes progress towards the fundamental understanding of heterogeneous and dynamic information systems and the way that we store and process massive data-sets.

Reliable large-scale data storage: Distributed storage systems for large clusters typically use replication to provide reliability. Recently, erasure codes have been used to reduce the large storage overhead of three-replicated systems. However, traditional erasure codes are associated with high repair cost that is often considered an unavoidable price to pay. In this thesis, we show how to overcome these limitations. We construct novel families of erasure codes that are optimal under various repair cost metrics, while achieving the best possible reliability. We show how these modern storage codes significantly outperform traditional erasure codes.

Low-rank approximations for large-scale data processing: A central goal in data analytics is extracting useful and interpretable information from massive data-sets. A challenge that arises from the distributed and large-scale nature of the data at hand, is having algorithms that are good in theory but can also scale up gracefully to large problem sizes. Using ideas from prior work, we develop a scalable low-rank optimization framework with provable guarantees for problems like the densest k-subgraph (DkS) and sparse PCA. Our experimental findings indicate that this low-rank framework can outperform the state-of-the art, by offering higher quality and more interpretable solutions, and by scaling up to problem inputs with billions of entries.

Table of Contents

Chapter 1

Introduction

This thesis revolves around two main topics. We start with discussing challenges and offering solutions for modern distributed data storage. Then, we continue on to large-scale data processing problems that are computationally intractable, and develop new scalable approximation algorithms that come with provable performance guarantees.

We start off with our data-storage Chapters 2, 3, and 4, where we study problems that arise in modern data storage applications. The new distributed nature of storage systems has given rise to new challenges, where off-the-shelf solutions for storing information reliably are no longer efficient. In this first part, we show how to reliably store information by developing and deploying modern erasure codes. Our erasure codes provably optimize several new performance metrics of interest.

In the second part of this thesis, in Chapters 5 and 6, we present a low-rank algorithmic framework for data analysis problems. The common objective in these problems is to find small components of large data-sets that offer significant information about the data. Using state-of-the-art low-rank solvers, we develop algorithms for these types of problems that have two key features: 1) they come with provable guarantees on the quality of the output solutions and 2) they are scalable to problems where the input (may it be a graph, a document-term matrix, etc.) can have billions of entries.

In the following, we give an overview of the specific problems that we study and outline our major contributions.

1.1 The Code Repair Problem in Large-scale Storage Systems

Traditional architectures for large-scale storage rely on systems that provide reliability through block replication. The major disadvantage of replication is

the large storage overhead. As the amount of stored data is growing faster than hardware infrastructure, this becomes a major data center cost bottleneck. *Erasure coding* techniques achieve higher data reliability with considerably smaller storage overhead [103]. For that reason various erasure codes are currently implemented and deployed in production storage clusters. Applications where coding techniques are being currently deployed include cloud storage systems like Windows Azure [51], big data analytics clusters (*e.g.*, the Facebook Analytics Hadoop cluster [88]), archival storage systems, and peer-to-peer storage systems like Cleversafe and Wuala.

It is now well understood that classical erasure codes (such as Reed-Solomon codes) are highly suboptimal for distributed storage settings [34]. For example, the Facebook analytics Hadoop cluster discussed in [88], deployed Reed-Solomon encoding for 8% of the stored data. This 8% of the stored data was reported to generate repair traffic that was approximately equal to 20% of the total network traffic. The fact that traditional erasure codes are not optimized for node repairs, is the main reason why they are not widely deployed in current storage systems.

Three major repair cost metrics have been identified in the recent literature: *i)* the number of bits communicated in the network, also known as the *repair-bandwidth* [17,34,77,81,95,99], *ii)* the number of bits read during each repair, *i.e.*, the *disk-I/O* [61, 99], and *iii)* more recently the number of nodes that participate in the repair process, also known as *repair locality*. Each of these metrics is more relevant for different systems and their fundamental limits are not completely understood.

In Fig. 1.1, we give an introductory sketch of the node repair problem. In Chapter 2, we give a gentle introduction to erasure codes and data replication, and introduce the repair problem more precisely.

Contribution 1: An open problem in the literature of distributed storage is that of constructing high-rate repair-communication optimal codes [34, 35]. These are erasure codes that minimize the number of bits communicated to repair a single node failure. In Chapter 3, we introduce the first explicit high-rate maximum distance separable (MDS) storage code with optimal repair communication. Our code construction is the first high data-rate code for distributed storage that is repair-communication optimal, resolving a 3 year standing open problem [77].

Contribution 2: In Chapter 4, we explore the repair metric of *locality*, which corresponds to the number of disk accesses required during a node repair. Under this

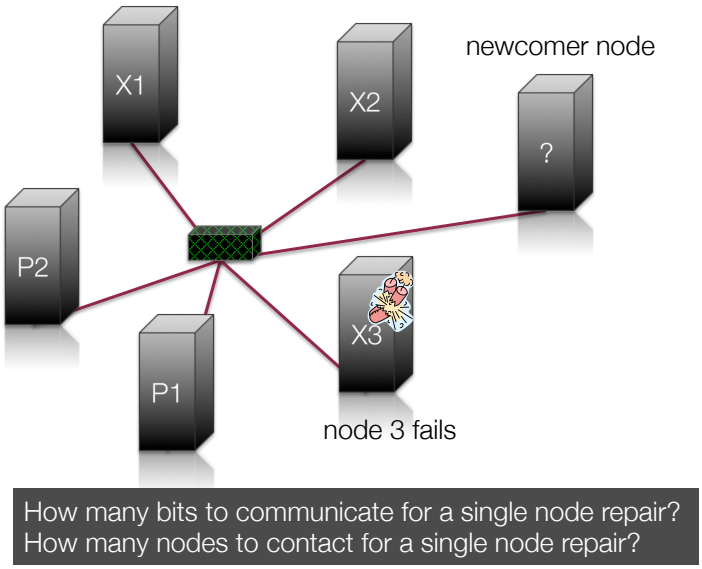


Figure 1.1: A sketch of a distributed storage system. A number of nodes, each containing different data blocks, is part of a larger storage network. In this network, single node failure events are not rare. Whenever a node is irreversibly damaged and its contents are lost, we ideally wish to regenerate what was lost in new nodes. A *newcomer node* joins the system and has to contact a number of surviving nodes, and then download a sufficiently large size of information to regenerate the lost contents. Regenerating this lost information is referred to as the *Node Repair Problem*, and two relevant questions are highlighted.

metric we characterize an information theoretic trade-off that binds together locality, code distance, and the storage capacity of each node. We study optimal *locally repairable codes* (LRCs), codes that are shown to achieve this trade-off.

During a singificant part of our LRC research, we collaborated with Facebook, and tested our codes in their clusters, as well as Amazon Ec2 [76,88]. Related locally repairable codes were recently deployed in production clusters in Microsoft Azure and currently ship in Windows 8.1 [51].

1.2 Low-rank approximations for intractable large-scale data analytics

The second direction of this thesis deals with large-scale data processing. The common underlying theme of this part is the study of quadratic optimization problems, subject to sparsity or other combinatorial constraints.

In the work of [60] and [10], a surprising result is shown: when the involved quadratic form matrix is positive semidefinite and constant rank, the maximization problem can be solved exactly, under various constraint sets that are combinatorial. This is achieved by sampling candidate solutions from a low-rank space that provably contains an optimal solution. However, as one would expect, most real data sets, yield matrices that are full rank. Interestingly, these matrices can often be approximated by low-rank surrogates with provably small approximation loss using different low-rank approximation methods ranging from simple Singular Value Decomposition (SVD) to more sophisticated techniques. We develop a general framework that combines low-rank matrix approximations with the low-rank solver of [10] to obtain provable solutions for hard combinatorial problems.

In this thesis, we apply this low-rank framework and analyze its theoretical performance on two problems: sparse principal component analysis (sparse PCA), and the densest k-subgraph problem. We obtain novel approximation guarantees for both problems, that depend on the spectrum of the involved matrices. In many cases, our framework outperforms the previous state of the art both theoretically and empirically. In the following, we give a brief description of sparse PCA and the densest k-subgraph problem.

1.2.1 Sparse PCA: finding a few features that explain large data sets

Principal Component Analysis (PCA) reduces data dimensionality by projecting it onto principal subspaces spanned by the leading eigenvectors of the sample covariance matrix. PCA is arguably the workhorse of high dimensional analysis; one of the most widely used algorithms with applications ranging from computer vision and document clustering to network anomaly detection, see *e.g.,* [54] and references therein.

One limitation of PCA is that the obtained eigenvectors typically have very few zero entries, *i.e.,* the principal components (PCs) are generally not sparse.

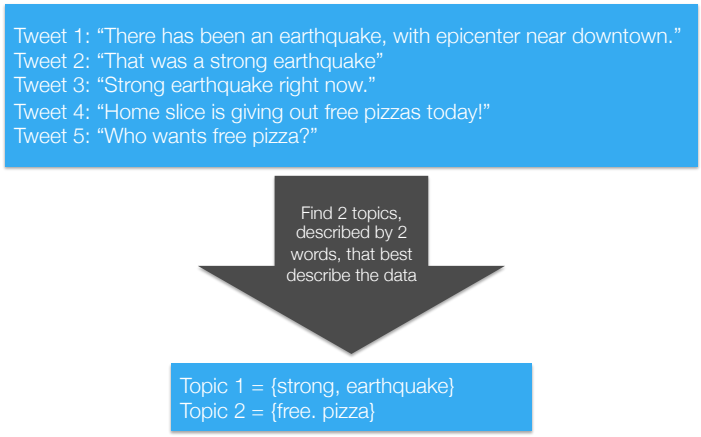


Figure 1.2: A simple proto-example of applying sparse PCA to a toy data-set. The input to the algorithm is a set of documents (in this case some Tweets), and the output is required to be a number of topics (here it is two), and each topic should be described by a small number of words (this is two again). The way these topics are computed, is such that they “best describe” the input data set. This measure of fitness (the objective that is maximized) in sparse PCA is the explained variance. Words that occur together frequently are expected to explain a lot of variance in the data set. Finding these words is experimentally shown to give interpretable results.

Sparsity in the PCs is desirable for data interpretability. Sparse PCA is a useful variant of PCA that enforces sparsity in the extracted components.

A well-known case where sparsity is desired for interpretability is document analysis [41, 79, 109]. Under the common “bag-of-words” model, document vectors and eigenvectors are supported on words. When the PCs are sparse, say they have only k non-zero values, this means that they are supported on only k words. If they are correctly extracted, the k words of the sparse PCs can be interpreted as well-separated topics. These vectors can be subsequently used to reduce the dimension, cluster data, and discover non-typical behavior. In Fig. 1.2, we give a sketch of this idea.

An open problem in the relevant literature [31], is having algorithms that come with *i)* provable performance guarantees and *ii)* are scalable to large data sets.

Contribution 3: In Chapter 5, we use the low-rank framework approach to obtain a novel approximation algorithm for sparse PCA. We obtain provable approximation guarantees that depend on the spectral profile of the matrix: the faster the eigenvalue decay, the better the quality of our approximation. For example, if the eigenvalues of the data-set matrix follow a power-law decay, we obtain a polynomial-time approximation algorithm for any desired accuracy.

Power-law spectral decays are particularly prevalent in real-world data-sets, as we show in our experiments. To the best of our knowledge, these guarantees are the tightest known for such a general family of data-sets. Experimental evaluations show that our scheme is not only good in theory, but is nearly optimal with respect to classic metrics of interest, while matching or outperforming previous algorithms in all tested data sets.

1.2.2 Finding dense subgraphs

The second problem that we study is the densest k-subgraph, a fundamental graph theory problem, with many applications including community detection. A sketch of the problem is presented in Fig. 1.3.

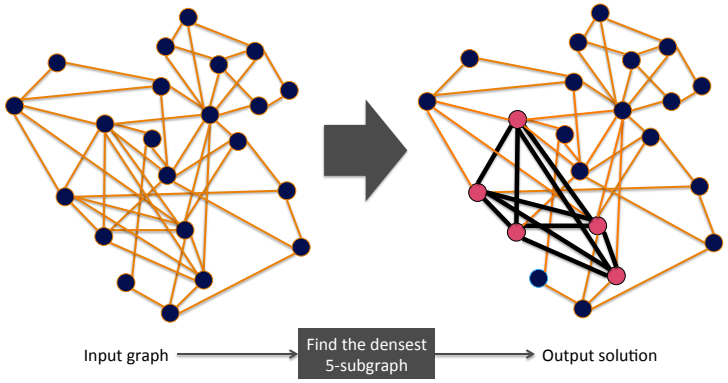


Figure 1.3: We are given a graph on n nodes, and a parameter k. The densest k-subgraph problem asks for the graph that sits on k vertices and contains the largest number of edges. In this example for $k = 5$ there exists a 5-clique (5 nodes that are all connected to each other). Unfortunately, finding the densest k-subgraph is computationally intractable for general graphs.

DkS is a notoriously hard problem. It is NP-hard by reduction to MAX-CLIQUE. Moreover, Khot showed in [62] that, under widely believed complexity theoretic assumptions, DkS cannot be approximated within an arbitrary constant factor.[1] The best known approximation ratio was $n^{1/3+\epsilon}$ (for some small ϵ) due to [40]. Recently, [15] introduced an algorithm that has an approximation ratio of $n^{1/4+\epsilon}$ and runs in time $n^{O(1/\epsilon)}$. Such results, where the approximation factor scales as a polynomial in the number of vertices, are too pessimistic for real-world applications. The resistance to better approximation despite the long history of the problem suggests that DkS is probably very hard in the worst case.

Contribution 4: In Chapter 6, we move beyond the worst case framework. We present a novel DkS algorithm based on the low-rank framework that has two key features: *i)* it comes with approximation guarantees that are surprisingly tight on real-world graphs and *ii*) it is fully parallelizable and can scale up to graphs with billions of edges. We illustrate this by tailoring our algorithm for a distributed implementation in the MapReduce framework and run experiments on Elastic MapReduce (EMR) on Amazon.

[1] approximation ratio ρ means that there exists an algorithm that produces in polynomial time a number A, such that $1 \leq \frac{\mathsf{opt}}{\mathsf{A}} \leq \rho$, where opt is the optimal density.

Part I

Large-Scale Data Storage

Chapter 2

The Code Repair Problem

Distributed storage systems for large scale applications typically use 3-way replication to provide reliability: for every bit stored, 2 more replicas are stored somewhere in the system. Recently, erasure codes have been used to reduce the large storage overhead of three-replicated systems [34, 103].

In this chapter, we give a very elementary introduction to erasure codes and compare them to replication. As we see, although classical codes outperform replication in terms of the achieved reliability for a given storage overhead, they are suboptimal under several repair metrics.

2.1 Replication vs. Erasure Codes: Reliability and Storage efficiency

The main difference of erasure codes compared to replication, is that instead of storing "raw" replicas of the file pieces, we can now use *functions* of the data, as show in Fig. 2.1. In that example, we have a file that we partition in two pieces, A and B. Then, in the case of replication we store these two pieces and a replica of each in four different nodes in a storage system. Instead of storing replicas, in the case of erasure codes we store linear equations of A and B. These linear equations are computed over an appropriate finite field.

A major advantage of erasure codes is their higher reliability compared to replication. For example, in Fig. 2.2, assume that the first and third blocks are erased. As shown in the figure, even after two block erasures, we can still recover all lost information using the remaining two blocks. However, in the case of 2-replication, even two erasures suffice to lose information.

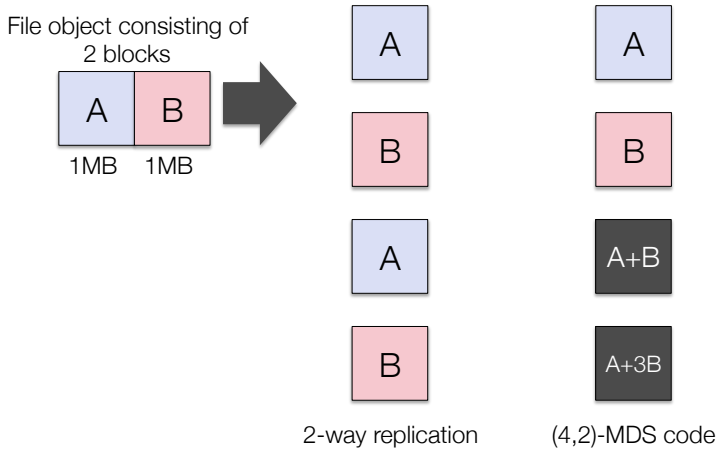


Figure 2.1: A file object of size 2MB is stored in 4 servers using either 2-replication or a $(4, 2)$-MDS code. When we introduce redundancy to our data and use erasure codes, we can store linear functions of the data (over a finite field). In the above $(4, 2)$-MDS example, instead of storing two replicas of A and B, we instead store two different linear combinations. A fundamental property of (n, k)-MDS codes is that any k of these linear equations have to be linearly independent. This allows for the best reliability possible, for the given number of parities (i.e., stored linear equations).

2.2 Replication vs. Erasure Codes: Repair Cost

Reed-Solomon codes are the standard design choice and their high repair cost is often considered an unavoidable price to pay for high storage efficiency and high reliability [34]. The repair cost corresponds to the resources spent to regenerate the contents of a system node, or a disk, when it fails. In Fig. 2.3, we give a toy example that exhibits this suboptimality.

A major open problem in this area has been the design of codes that *i)* are repair efficient and *ii)* achieve arbitrarily high data rates. In the first part, Chapters 3 and 4, we show how we can overcome this limitation of traditional erasure codes by developing codes that are optimized for two repair metrics discussed in the following.

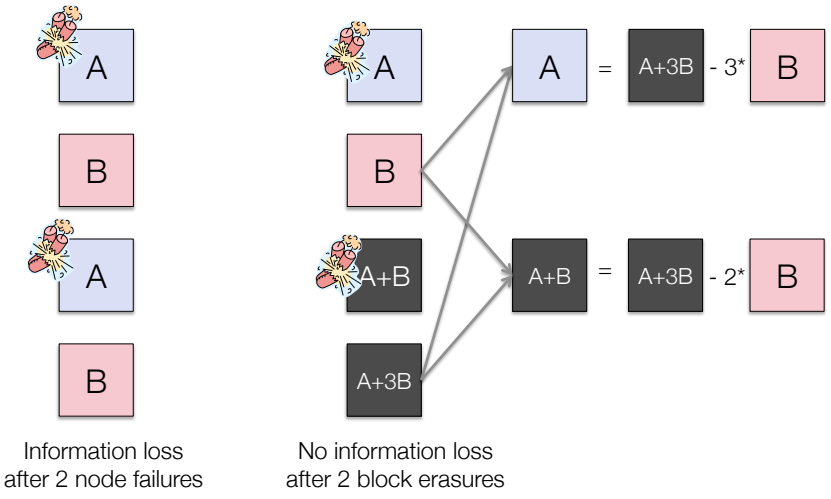


Figure 2.2: A file object of size 2MB is stored in 4 servers using either 2-replication or a $(4, 2)$-MDS code. The above is a simple example of how coding can be better than replication, in terms of the reliability that it offers. For our (4,2) code, any two erasures can be tolerated. For example we show how to reconstruct what was lost, when the first and third nodes fail. On the other hard, replication suffers from data loss when the same two blocks are erased. Hence, for the same data overhead, even this toy example shows how codes can be better than replication.

2.2.1 Repair communication: how many bits to communicate?

An open problem in the literature of distributed storage is that of constructing high-rate repair-communication optimal codes [34, 35]. In [34] a fundamental question was partially answered:

" how many bits do we need to communicate to repair a single node failure?"

The answer to the above question came as an information theoretic trade-off in [34] that specified precisely the minimum number of bits required for communication during node repairs. Codes that achieve this bound are referred to as *repair bandwidth optimal*. These are erasure codes that minimize the number of bits communicated to repair a single node failure. The construction of high-rate repair optimal codes has since been an open problem.

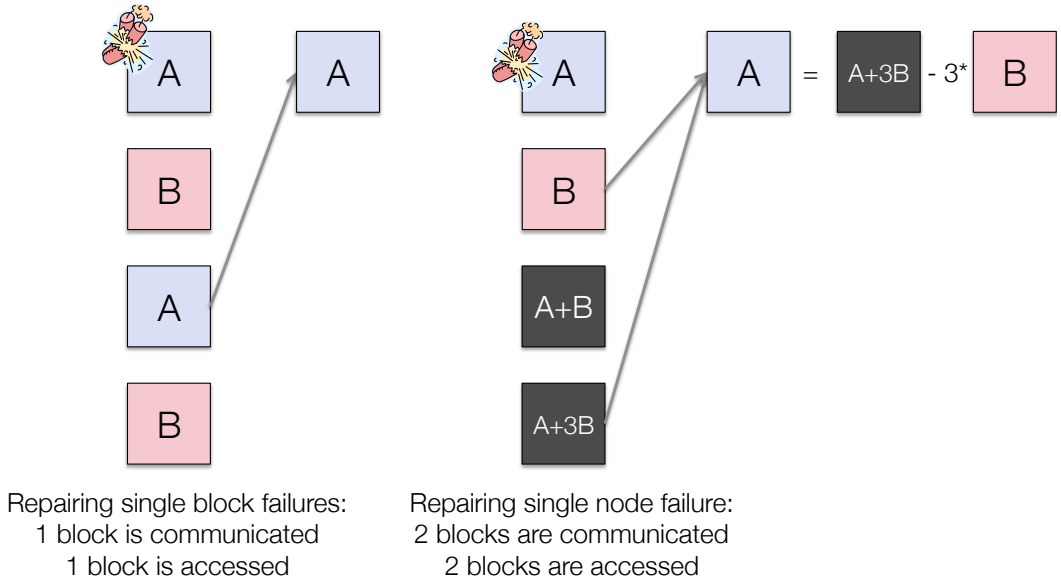


Figure 2.3: A file object of size 2MB is stored in 4 servers using either 2-replication or a $(4, 2)$-MDS code. A main issue of traditional erasure codes is their high repair cost. In the above example, observe that only a single block (i.e., A) is lost, and it is of size 1MB. However, to repair it we need to download 2 blocks of total size 2MB, and contact 2 remaining nodes. On the other hard, when replication is used, we simply need to contact a single node and download exactly one block. The main question that we are interested in is: can we construct codes that are repair efficient?

Towards this direction, in Chapter 3 we develop and exploit a precise linear algebraic theory that quantifies this repair cost [77]. Using this theory, we show a surprising algebraic connection between two fundamentally different metrics in two unrelated setups: the repair cost of a disk failure and the data rate of a wireless communication channel. Using this connection, we are able to provide the first scheme for finite symbol interference alignment, an open problem in wireless communication channels. Our finite interference alignment method, uses the intricate structure of Hadamard matrices that is a key in the design of repair optimal codes.

Our code construction is the first high data-rate code for distributed storage that was repair-communication optimal, resolving a 3 year standing open problem [77].

2.2.2 Repair locality: how many nodes to talk to?

In Chapter 4, we explore the repair metric of *locality*, which corresponds to the number of disk accesses required during a node repair. We give an answer to the following fundamental question.

"how many nodes do we need to contact to repair a single node failure?"

As with the metric of repair communication, the answer to the above question will also come in the form of a fundamental trade-off. We characterize an information theoretic trade-off that binds together locality (the number of nodes that participate in the repair process), code distance, and the storage capacity of each node.

We then study optimal *locally repairable codes* (LRCs), codes that are shown to achieve this trade-off. The achievability proof uses a locality aware flow-graph gadget which leads to a randomized code construction. Finally, we present an optimal and explicit LRC construction that achieves arbitrarily high data-rates. Our locality optimal code is based on simple combinations of Reed-Solomon coded blocks.

In subsequent research, we implemented our constructions in the Hadoop DFS and test in real experiments against the current coded version of Hadoop, where we observe surprising performance gains [88].

Chapter 3

Repair Optimal Erasure Codes through Hadamard Designs

Designing high-rate maximum-distance separable (MDS) codes that achieve the optimum repair communication has been a well-known open problem. In this chapter, we use Hadamard matrices to construct the first explicit 2-parity MDS storage code with optimal repair properties *for all* single node failures, including the parities.

Our construction relies on a novel method of achieving *perfect* interference alignment over finite fields with a finite number of symbol extensions. We generalize this construction to design m-parity MDS codes that achieve the optimum repair communication for single systematic node failures.[1]

3.1 Introduction

Currently, the most well-understood repair metric is that of repair bandwidth. In this work we are particularly interested in constructing (n, k)-MDS storage codes that are optimal with respect to the repair bandwidth. The information theoretic bounds for repair bandwidth were specified in [34] and shown to be asymptotically, and in some cases exactly tight for all values of n, k in a series of recent papers [20,35,81,82,89,96]. Beyond MDS codes, [34] demonstrated a tradeoff between storage and repair bandwidth, and code constructions for other points of this tradeoff are under active investigation, see e.g. [35,81,90,92]. On this tradeoff, the minimum storage point is achieved by MDS erasure codes with optimal repair, also known as Minimum Storage Regenerating (MSR) codes.

For code rates $k/n \leq 1/2$, explicit MSR codes were designed by Shah *et al.* [89], Rashmi *et al.* [81], and Suh *et al.* [95]. For the high-rate regime however,

[1] Contributions Statement: Most parts of this chapter appear in [77]. Prof. Alex Dimakis supervised the project and all coauthors had equal contribution to this work.

with the exception of the special cases where $k = 2, 3,$ [27, 89, 95, 104], the only known complete constructions [20, 96] require arbitrarily large file sizes (symbol extensions) and field order. These constructions use the symbol extension interference alignment technique of [21] to establish that there exist MDS storage codes, that come arbitrarily close to (but do not exactly match) the information theoretic lower bound of the repair bandwidth for all n, k. These asymptotic constructions are impractical due to the arbitrarily large finite field order and the fast growing file size that is required, even for small values of n and k.

Our Contribution: We introduce the first explicit high-rate $(k+2, k)$-MDS storage code with optimal repair communication. Our storage code exploits fundamental properties of Hadamard designs and perfect interference alignment instances that can be understood through the use of a lattice representation of the symbol extension technique of Cadambe *et al.* [20, 21, 96].

Independently of this work, there has recently been a substantial progress in designing high-rate explicit MSR codes. Tamo et al. [99] and Cadambe et al. [17] designed MDS codes for any (n, k) parameters that have optimal repair for the systematic nodes. In fact, while several code constructions exist for repairing systematic nodes optimally [17, 99], the existence of codes which can optimally repair parity nodes as well remained an open problem. The advantage of our work is that all n nodes are optimally repaired and the disadvantage is that our construction is currently only optimal for $n - k = 2$. In parallel to and independently from our results, wang2011codes et al. [102] have constructed codes which are optimal for the repair of all nodes. The construction of [102] is different from ours, though exploration of underlying connections is a possible direction for future work.

Our key technical contribution is a scheme that achieves *perfect* interference alignment with a finite number of extensions that we present in Section 3.3. This was developed in [78] and used in a 2 parity storage code with optimal repair for k nodes and near optimal repair of 2 nodes, that can handle any single node failure. We use a combinatorial view of different interference alignment schemes using a framework we call *dots-on-a-lattice*. Hadamard matrices are shown to be crucial in achieving finite perfect alignment while ensuring the full-rank of desired subspaces. In Sections 3.5 and 3.6, we prove the repair bandwidth optimality of our code construction. In Section 3.7, we give explicit conditions on the MDS property

of the code and show that a finite field of order greater than or equal to $2k+3$ suffices to satisfy them.

Finally, in Section 3.8, we present m-parity MDS code constructions based on Hadamard designs that achieve optimal repair for systematic node failures, but are suboptimal for parity node repairs. The MDS property for these designs is probabilistically guaranteed by choosing random constants multiplying the coding matrices.[2] In Section 3.9, we show that our m-parity codes are equivalent to codes that involve permutation matrices, in the manner of [99] and [17], under a similarity transformation of the coding matrices.

3.2 MDS Storage Codes with 2 Parity Nodes

In this section, we consider the code repair problem for MDS storage codes with 2 parity nodes. After we lay down the model for repair, we continue with introducing our code construction.

Let a file of size $M = kN$ denoted by the vector $\mathbf{f} \in \mathbb{F}_q^{kN}$ be partitioned in k parts $\mathbf{f} = \left[\mathbf{f}_1^T \dots \mathbf{f}_k^T\right]^T$, each of size N, where N denotes the *subpacketization* factor[3], $\frac{N}{2} \in \mathbb{N}^*$.[4] We encode $\mathbf{f}$ using an $(n = k+2, k)$ code and store it across k systematic and 2 parity storage units, each having storage capacity $\frac{M}{k} = N$. Hence, the effective coding rate is $R = \frac{k}{k+2}$. We require that our code is MDS, i.e., the encoded storage array is resilient to up to any 2 node erasures. A storage code has the MDS property when any collection of k storage nodes can reconstruct the file $\mathbf{f}$.

In Fig. 4.4, we provide a generic representation of a 2-parity MDS encoded storage array. The first k storage nodes store the systematic file parts. Without loss of generality, the first parity stores the sum of all k systematic parts $\mathbf{f}_1 + \ldots + \mathbf{f}_k$ and the second parity stores a linear combination of them $\mathbf{A}_1^T\mathbf{f}_1 + \ldots + \mathbf{A}_k^T\mathbf{f}_k$.[5] Here, $\mathbf{A}_i$

[2] All codes presented in this work are for the case where during a node repair all $d = n-1$ surviving nodes participate in the process. Generalizations where only subsets of the surviving nodes participate in repair have been pursued for special cases of (n,k) in literature (See for example, [2, 34, 35, 38, 81, 90])

[3] A larger file can be cut into pieces of size M where coding is performed independently on these pieces. If splitting the file in smaller pieces leaves the last piece having size less than M, then we can zero-pad it and encode it without storing the zero-padded blocks.

[4] $\mathbb{F}_q$ denotes the finite field over which all operation are performed.

[5] The MDS property and the repair bandwidth of a code are invariant under a change of basis [81].

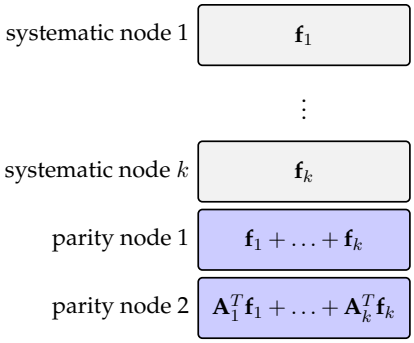


Figure 3.1: A $(k+2, k)$ encoded storage array.

denotes an $N \times N$ matrix of coding coefficients used by the second parity node to "scale and mix" the contents of the ith file piece $\mathbf{f}_i$, $i \in [k]$, where $[N] = \{1, \ldots, N\}$. This representation is a systematic one: k nodes store uncoded file pieces and each of the 2 parities stores a linear combination of the k file parts.

When a node of the $(k+2, k)$ encoded array fails, a *newcomer* node joins the system, downloads sufficient information from the remaining $d = n - 1 = k + 1$ nodes and regenerates what was lost. Hence, when a failure occurs, the *Code Repair* process is initiated to exactly regenerate the lost coded data in a *newcomer* storage component. See Fig. 2, for a sketch of the repair of a generic $(4, 2)$-MDS code.

We now consider that a systematic node $i \in [k]$ fails. Then, a newcomer storage node joins the storage network, it connects to the remaining nodes, and has to download sufficient data to reconstruct $\mathbf{f}_i$. Observe that the lost systematic part $\mathbf{f}_i$ exists *only* as a term of a linear combination at each parity node, as seen in Fig. 1. Since $\mathbf{f}_i$ has size N, to (linearly) regenerate it, the newcomer has to download from the parity nodes at least N linearly independent equations. Assuming that it downloads the same amount of data from both parities, the downloaded contents can be represented as a stack of N equations

$$\begin{aligned}\begin{bmatrix} \mathbf{p}_i^{(1)} \\ \mathbf{p}_i^{(2)} \end{bmatrix} &\triangleq \begin{bmatrix} \left(\mathbf{V}_i^{(1)}\right)^T \mathbf{f}_1 + \ldots + \left(\mathbf{V}_i^{(1)}\right)^T \mathbf{f}_k \\ \left(\mathbf{V}_i^{(2)}\right)^T \mathbf{A}_1^T \mathbf{f}_1 + \ldots + \left(\mathbf{V}_i^{(2)}\right)^T \mathbf{A}_k^T \mathbf{f}_k \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} \left(\mathbf{V}_i^{(1)}\right)^T \\ \left(\mathbf{A}_i \mathbf{V}_i^{(2)}\right)^T \end{bmatrix} \mathbf{f}_i}_{\text{useful data}} + \sum_{s=1, s\neq i}^{k} \underbrace{\begin{bmatrix} \left(\mathbf{V}_i^{(1)}\right)^T \\ \left(\mathbf{A}_s \mathbf{V}_i^{(2)}\right)^T \end{bmatrix} \mathbf{f}_s}_{\text{interference by } \mathbf{f}_s}, \end{aligned} \tag{3.1}$$

where $\mathbf{p}_i^{(1)}, \mathbf{p}_i^{(2)} \in \mathbb{F}_q^{\frac{N}{2}}$ are the equations downloaded from the first and second parity node, respectively, and $\mathbf{V}_i^{(1)}, \mathbf{V}_i^{(2)} \in \mathbb{F}_q^{N \times \frac{N}{2}}$ are *repair matrices*. Each repair matrix is used to mix the N parity equations to form $\frac{N}{2}$ "repair equations." Retrieving $\mathbf{f}_i$ from Eq. (3.1) is equivalent to solving an underdetermined set of N equations in the kN unknowns of $\mathbf{f}$, with respect to the N desired unknowns of $\mathbf{f}_i$. However, this is not possible due to $k-1$ additive *interference* components in the received equations generated by the undesired unknowns $\mathbf{f}_s$, $s \neq i$, as noted in Eq. (3.1). These $k-1$ interference terms are combined with the desired data and need to be canceled. Therefore, the newcomer needs to download additional data from the remaining $k-1$ systematic nodes. These new equations will be used to replicate and cancel the interference terms from the downloaded parity equations.

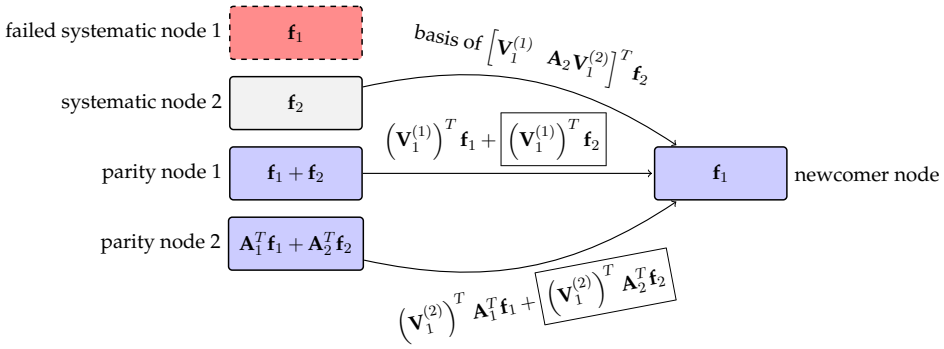


Figure 3.2: The code-repair problem for a $(4, 2)$ code. Here, $\mathbf{A}_1$ and $\mathbf{A}_2$ are $N \times N$ coding matrices and each $\mathbf{f}_1$ and $\mathbf{f}_2$ has size N. Let the first node fail. Then, a newcomer node joins the system and downloads data from the 3 remaining nodes to regenerate $\mathbf{f}_1$. The useful information is mixed with the undesired part $\mathbf{f}_2$ in both data blocks downloaded from the two parities. The interference parts appear inside a box. To retrieve $\mathbf{f}_1$, the interference terms need to be erased. For that a basis of these terms needs to be downloaded by systematic node 2. Then, the newcomer can erase the interference. Note that for succesful regeneration of $\mathbf{f}_1$ we also require that the matrix $[\mathbf{V}_1^{(1)} \quad \mathbf{A}_2\mathbf{V}_1^{(2)}]$ has full-rank N.

To cancel a single interference term of Eq. (3.1) that has size N, it suffices to download a basis of equations that can generate it. For example, to erase the interference component generated by the file part $\mathbf{f}_s$, the newcomer needs to connect to systematic node s and download a number of linear equations in $\mathbf{f}_s$ that can

generate it. This number is equal to

$$\text{rank}\left(\begin{bmatrix} \left(\mathbf{V}_i^{(1)}\right)^T \\ \left(\mathbf{A}_s\mathbf{V}_i^{(2)}\right)^T \end{bmatrix}\right)$$

which satisfies

$$\frac{N}{2} \leq \text{rank}\left(\begin{bmatrix} \left(\mathbf{V}_i^{(1)}\right)^T \\ \left(\mathbf{A}_s\mathbf{V}_i^{(2)}\right)^T \end{bmatrix}\right) \leq N. \tag{3.2}$$

Hence, in terms of downloaded equations, this is exactly the repair-bandwidth required to delete the interference term caused by $\mathbf{f}_s$. The lower bound in Eq. (3.2) comes from the fact that $\frac{N}{2}$ linearly independent equations need to be downloaded from each of the parities, thus

$$\text{rank}\left(\mathbf{V}_i^{(1)}\right) = \text{rank}\left(\mathbf{V}_i^{(2)}\right) = \frac{N}{2}.$$

To erase all interference terms, the newcomer needs to download an aggregate of

$$\sum_{s=1,s\neq i}^{k} \text{rank}([\mathbf{V}_i^{(1)} \;\; \mathbf{A}_s\mathbf{V}_i^{(2)}])$$

equations from the remaining $k-1$ systematic nodes. We note that posterior to erasing interference terms, we require that the remaining N equations in the N unknowns of $\mathbf{f}_i$ are a full-rank system N, i.e., $\text{rank}([\mathbf{V}_i^{(1)} \;\; \mathbf{A}_i\mathbf{V}_i^{(2)}]) = N$. Again, please see Fig. 2 for a generic example of a $(4,2)$-MDS storage code repair instance. Hence, we can state the repair problem of a systematic node i as a rank constrained, rank minimization one, performed over $\mathbb{F}_q$.

$$\begin{aligned} \mathcal{R}_i\text{:} \min_{\mathbf{V}_i^{(1)},\mathbf{V}_i^{(2)}} & \sum_{s=1,s\neq i}^{k} \text{rank}\left(\left[\mathbf{V}_i^{(1)} \;\; \mathbf{A}_s\mathbf{V}_i^{(2)}\right]\right) \\ \text{s.t.: } & \text{rank}\left(\left[\mathbf{V}_i^{(1)} \;\; \mathbf{A}_i\mathbf{V}_i^{(2)}\right]\right) = N. \end{aligned} \tag{3.3}$$

Remark 1. *From [34] it is known that the theoretical minimum repair bandwidth, for any single node repair of an optimal (linear or nonlinear) $(k+2,k)$-MDS code is exactly $\frac{1}{2}$ times the number of remaining equations in the system, i.e., $(k+1)\frac{N}{2}$. This bound is proven using cut-set bounds on an infinite flow graph. Here, we provide an interpretation of this bound in terms of linear codes by calculating the minimum possible sum of ranks in*

$\mathcal{R}_i$. Since each repair matrix has to have full column rank of $\frac{N}{2}$ to be a feasible solution, the minimum rank each interference term can possibly have is $\frac{N}{2}$. This aggregates in a minimum repair bandwidth of $N + (k-1)\frac{N}{2} = (k+1)\frac{N}{2}$ repair equations. Interestingly, linear codes suffice to asymptotically achieve this bound [20], [96].

Although the theoretical minimum repair bandwidth has been established in the literature and asymptotically optimal schemes that achieve it with a finite block length have been constructed, high-rate MDS codes that achieve it has been a challenging open problem. The difficulty in designing optimal MDS storage codes lies in a threefold requirement: *i)* the code has to satisfy the MDS property, *ii)* systematic nodes of the code have to be optimally repaired, and *iii)* parity nodes of the code have to be optimally repaired. Currently, there exist MDS codes for the low-rate regime, i.e., $\frac{k}{n} \leq \frac{1}{2}$, for which all nodes can be optimally repaired [81, 89, 95]. For the high data rate regime, Tamo *et al.* [99] and Cadambe *et al.* [17] presented the first MDS codes where any systematic node failure can be optimally repaired. Prior to this work, there did not exist MDS storage codes of arbitrarily high-rate that can optimal repair *any* node.

In the following, we present an explicit, high-rate $(k+2, k)$-MDS storage code. Our code achieves the minimum repair bandwidth bound for the repair of *any* single systematic or parity node failure. Before we proceed with the construction itself, we will state the intuition behind our code constructions and the tools that we use. Motivated by asymptotic alignment schemes, we use similar concepts induced by a combinatorial explanation of interference alignment in terms of dots on lattices. In contrast to the asymptotic interference alignment codes of [20] and [96], here, instead of letting randomness choose the coding matrices, we select particular constructions based on Hadamard matrices that achieve *exact* interference alignment in finite symbol extensions. In Section 3.5, we prove the optimal repair of systematic nodes, in Section 3.6 we show the optimal repair of parity nodes, and in Section 3.7 we state explicit conditions for the MDS property.

3.3 Dots-on-a-lattice and Hadamard Designs

In this section, we simplify our ultimate goal of finding codes with minimal repair bandwidth defined in problem $\mathcal{R}_i$ of Section 6.2. On simplifying the problem at hand, we will explain our Hadamard matrix based design which lies at the heart

of the code constructions described in Eq. (3.20). Consider $\mathcal{R}_i$ and let us say that node $i = 1$ fails. Then, we would like to repair it by downloading the minimum amount of information, i.e., we aim to minimize the repair bandwidth which is equivalent to minimizing

$$\sum_{s=2}^{k} \text{rank}\left([\mathbf{A}_s\mathbf{V}_1 \;\; \mathbf{V}_1]\right). \tag{3.4}$$

Observe that here, and in the construction that follows, when we repair a systematic node, we use the same repair matrix $\mathbf{V}_i$ for all parities.

To minimize the repair bandwidth required to regenerate node $i = 1$, we would like to maximize the overlap (alignment) of $\mathbf{V}_1$ and $\mathbf{A}_s\mathbf{V}_1$ for $s \neq 1$. Ideally, we would like to have all the columns of $\mathbf{A}_s\mathbf{V}_1$ to lie in the column space of $\mathbf{V}_1$, so that the rank of $[\mathbf{A}_s\mathbf{V}_1 \;\; \mathbf{V}_1]$ is as small as the rank of $\mathbf{V}_1$. In other words, we would like $\mathbf{V}_1$ to be an *invariant subspace* of $\mathbf{A}_s,\ s \in [k]\backslash 1$

$$\text{colspan}(\mathbf{V}_1) = \text{colspan}(\mathbf{A}_2\mathbf{V}_1) = \ldots = \text{colspan}(\mathbf{A}_k\mathbf{V}_1) \tag{3.5}$$

so that $\mathbf{V}_1$ completely overlaps with all $\mathbf{A}_s\mathbf{V}_1$, for $s \neq 1$, as desired. This idea is central to our constructions. In this section, we pursue a simpler problem whose solution captures the aforementioned idea. We attempt to find *two* matrices which we denote by $\mathbf{T}_1$ and $\mathbf{T}_2$ and a matrix $\mathbf{V}$ which is invariant to both $\mathbf{T}_1, \mathbf{T}_2$.

We now consider two arbitrary $N \times N$ full-rank matrices $\mathbf{T}_1$ and $\mathbf{T}_2$ that commute. We wish to construct a full-rank matrix $\mathbf{V}$, with at most $\frac{N}{2}$ columns, such that the span of $\mathbf{T}_1\mathbf{V}$ aligns as much as possible with the span of $\mathbf{T}_2\mathbf{V}$: we have to pick $\mathbf{V}$ such that it minimizes the dimensions of the union of the two spans, that is the rank of $[\mathbf{T}_1\mathbf{V} \;\; \mathbf{T}_2\mathbf{V}]$. How can we construct such a matrix? Assume that we start with one vector with nonzero entries, i.e., $\mathbf{V} = \mathbf{w}$, and for simplicity we let it be the all-ones vector. Then in the general case, $\mathbf{T}_1\mathbf{w}$ and $\mathbf{T}_2\mathbf{w}$ have zero intersection which is not desired. However, we can augment $\mathbf{V}$ such that it has as columns the elements of the set $\{\mathbf{w}, \mathbf{T}_1\mathbf{w}, \mathbf{T}_2\mathbf{w}, \mathbf{T}_1\mathbf{T}_2\mathbf{w}\}$. This idea of augmenting the set $\mathbf{V}$ in this manner to increase the overlap is presented in [21]. Observe that each vector $\mathbf{T}_1^{x_1}\mathbf{T}_2^{x_2}\mathbf{w}$ of $\mathbf{V}$ can be represented by the power tuple (x_1, x_2). This helps us visualize $\mathbf{V}$ as a set of dots on the 2-dimensional integer lattice as shown in Fig. 3.3.

For this new selection of $\mathbf{V}$, we have

$$\mathbf{T}_1\mathbf{V} = \begin{bmatrix}\mathbf{T}_1\mathbf{w} & \mathbf{T}_1^2\mathbf{w} & \mathbf{T}_1\mathbf{T}_2\mathbf{w} & \mathbf{T}_1^2\mathbf{T}_2\mathbf{w}\end{bmatrix} \tag{3.6}$$

$$\text{and } \mathbf{T}_2\mathbf{V} = \begin{bmatrix}\mathbf{T}_2\mathbf{w} & \mathbf{T}_2\mathbf{T}_1\mathbf{w} & \mathbf{T}_2^2\mathbf{w} & \mathbf{T}_1\mathbf{T}_2^2\mathbf{w}\end{bmatrix}. \tag{3.7}$$

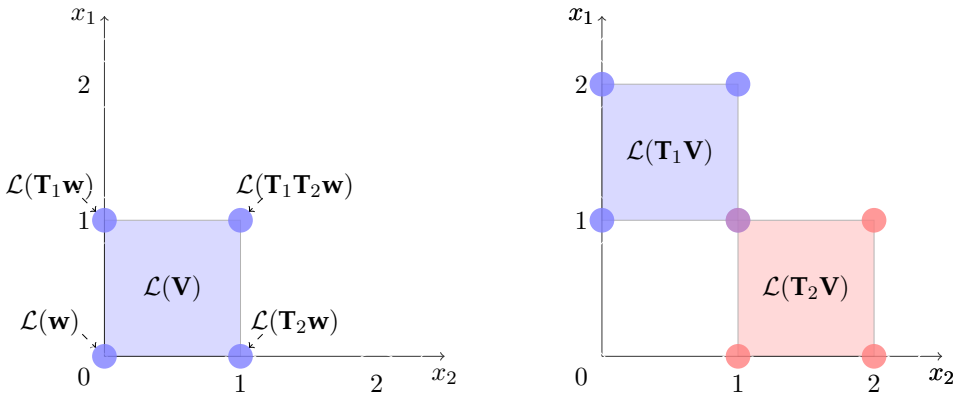


Figure 3.3: We use an $\mathcal{L}$ map from vectors generated as $\mathbf{T}_1^{x_1}\mathbf{T}_2^{x_2}\mathbf{w}$ to (x_1, x_2) lattice points on $\mathbb{Z}^2$. We first represent $\mathbf{V}$ as dots-on-a-lattice, where $\mathbf{V} = [\mathbf{w} \;\; \mathbf{T}_1\mathbf{w} \;\; \mathbf{T}_2\mathbf{w} \;\; \mathbf{T}_1\mathbf{T}_2\mathbf{w}]$ and $\mathcal{L}(\mathbf{V}) = \{(0,0),(1,0),(0,1),(1,1)\}$. We also depict the representation of the matrix $[\mathbf{T}_1\mathbf{V} \;\; \mathbf{T}_2\mathbf{V}]$ as dots-on-a-lattice, i.e., $\mathcal{L}([\mathbf{T}_1\mathbf{V} \;\; \mathbf{T}_2\mathbf{V}]) = \mathcal{L}(\mathbf{T}_1\mathbf{V}) \cup \mathcal{L}(\mathbf{T}_2\mathbf{V}) = \{(1,0),(2,0),(1,1),(2,1),(0,1),(1,1),(0,2),(1,2)\}$. Observe that if there existed a "wrap-around" cyclic property on the $\mathbf{T}_1, \mathbf{T}_2$ matrices, then the sets of dots $\mathcal{L}(\mathbf{T}_2\mathbf{V})$ and $\mathcal{L}(\mathbf{T}_2\mathbf{V})$ could potentially overlap.

The intersection of the spans of these two matrices is now nonzero: the matrix $[\mathbf{T}_1\mathbf{V} \;\; \mathbf{T}_2\mathbf{V}]$ has rank 7 instead of the maximum possible of 8. This happens because the vector $\mathbf{T}_1\mathbf{T}_2\mathbf{w}$ is repeated in both matrices $\mathbf{T}_1\mathbf{V}$ and $\mathbf{T}_2\mathbf{V}$. In Fig. 3.3, we illustrate this concatenation, in terms of dots on $\mathbb{Z}^2$, where the intersection between the two spans is manifested as an overlap of dots. Observe how matrix multiplication of $\mathbf{T}_1$ and $\mathbf{T}_2$ with the vectors in $\mathbf{V}$ is pronounced through the dots representation: the dots representations of $\mathbf{T}_1\mathbf{V}$ and $\mathbf{T}_2\mathbf{V}$ matrices are shifted versions of $\mathcal{L}(\mathbf{V})$ along the x_1 and x_2 axes.

However, the $\mathbf{T}_i$ matrices (which in our case are coding matrices) are free to design under some specific constraints (which in our case is the MDS property of the code). Therefore, we can try to construct explicit matrices such that

$$\text{span}(\mathbf{T}_1\mathbf{V}) = \text{span}(\mathbf{T}_2\mathbf{V}), \tag{3.8}$$

as we required in Eq. (3.5). Interestingly, perfect and finite symbol extension interference alignment instances are possible when we enforce the dots representation

of the $\mathbf{V}$ matrix to wrap-around itself and have a cyclic property. This wrap-around property is crucial in enabling perfect alignment of spaces. We will see that this property is obtained when the elements of the matrices are m^{th} roots of unity, i.e.,

$$\mathbf{T}_i^m = \mathbf{I}_N. \tag{3.9}$$

To see that, we formally state the dots-on-a-lattice representation. Let a map $\mathcal{L}$ from a matrix with r columns, each generated as $\mathbf{T}_1^{x_1}\mathbf{T}_2^{x_2}\mathbf{w}$, to a set of r points, such that the column $\mathbf{T}_1^{x_1}\mathbf{T}_2^{x_2}\mathbf{w}$ maps to the point (x_1, x_2). Then, we have for $\mathbf{V}$

$$\mathcal{L}(\mathbf{V}) \stackrel{\triangle}{=} \{x_1\mathbf{e}_1 + x_2\mathbf{e}_2;\ x_1, x_2 \in \{0, \ldots, m-1\}\}, \tag{3.10}$$

where $\mathbf{e}_i$ is the i-th column of the identity matrix. Using this representation, the products $\mathbf{T}_1\mathbf{V}$ and $\mathbf{T}_2\mathbf{V}$ map to

$$\mathcal{L}(\mathbf{T}_1\mathbf{V}) = \left\{(x_1+1)\mathbf{e}_1 + x_2\mathbf{e}_2 : x_1, x_2 \in \{0, \ldots, m-1\}\right\}$$
$$\text{and } \mathcal{L}(\mathbf{T}_2\mathbf{V}) = \left\{x_1\mathbf{e}_1 + (x_2+1)\mathbf{e}_2 : x_1, x_2 \in \{0, \ldots, m-1\}\right\}.$$

For perfect alignment, we have to design the $\mathbf{T}_i$ matrices such that $\mathcal{L}(\mathbf{T}_1\mathbf{V}) = \mathcal{L}(\mathbf{T}_2\mathbf{V})$. A sufficient set of conditions for perfect span intersection is that the power tuples of $\mathbf{V}$, $\mathbf{T}_1\mathbf{V}$, and $\mathbf{T}_2\mathbf{V}$ perfectly intersect. Consider for example the following condition: $\mathcal{L}(\mathbf{T}_1\mathbf{V}) = \mathcal{L}(\mathbf{V})$, i.e.,

$$\left\{(x_1+1)\mathbf{e}_1 + x_2\mathbf{e}_2 : x_1, x_2 \in [m]\right\} = \left\{x_1\mathbf{e}_1 + x_2\mathbf{e}_2 : x_1, x_2 \in [m]\right\}.$$

The above condition means that when we multiply $\mathbf{T}_i^{m-1}$ with $\mathbf{T}_i$, the new product maps to $\mathbf{I}$, i.e., the $\mathbf{T}_i^{x_i}$ has a cyclic "power periodicity" of m. Hence, the aforementioned perfect alignment conditions are satisfied when the matrix powers "wrap around" upon reaching their modulus, m, i.e., when the additions $x_1 + 1$ and $x_2 + 1$ are performed modulo m. This wrap-around property is obtained when the $\mathbf{T}_1$ and $\mathbf{T}_2$ are diagonals of m^{th} roots of unity

$$\mathbf{T}_1^m = \mathbf{T}_1^0 = \mathbf{T}_2^m = \mathbf{T}_2^0 = \mathbf{I}_N. \tag{3.11}$$

Remark 2. *Observe that for diagonal matrices $\mathbf{T}_1, \mathbf{T}_2$, where the diagonal elements are m-th roots of unity, the operator $\mathcal{L}$ defines an isomorphism, between the elements of the group $\mathbf{T}_1^{x_1}\mathbf{T}_2^{x_2}\mathbf{w}$, under the left-hand-side matrix product operation with $\mathbf{T}_1, \mathbf{T}_2$ matrices, and the elements of $\mathbb{F}_m^2$ under addition.*

Arbitrary selection of $\mathbf{T}_1, \mathbf{T}_2$ with diagonals as roots of unity is not sufficient to ensure the full-rank property of $\mathbf{V}$. To hint on a general procedure which outputs "good" matrices, we see an example where we tune our parameters such that $\mathbf{V}$ has orthogonal columns. We would like to note that although we consider orthogonality at this time, linear independence of the columns of $\mathbf{V}$ always suffices. Let us briefly consider the case where $m = 2$ and $N = 2^3$, for which we choose

$$\mathbf{T}_1 = \text{diag}\left(\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}\right) \text{ and } \mathbf{T}_2 = \text{diag}\left(\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}\right). \tag{3.12}$$

For these matrices, $\mathbf{V}$ has $m^2 = 4$ orthogonal columns

$$\mathbf{V} = [\mathbf{w} \quad \mathbf{T}_1\mathbf{w} \quad \mathbf{T}_2\mathbf{w} \quad \mathbf{T}_1\mathbf{T}_2\mathbf{w}] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{3.13}$$

and $\mathbf{T}_1\mathbf{V} = [\mathbf{T}_1\mathbf{w} \quad \mathbf{w} \quad \mathbf{T}_1\mathbf{T}_2\mathbf{w} \quad \mathbf{T}_2\mathbf{w}]$, $\mathbf{T}_2\mathbf{V} = [\mathbf{T}_2\mathbf{w} \quad \mathbf{T}_1\mathbf{T}_2\mathbf{w} \quad \mathbf{w} \quad \mathbf{T}_1\mathbf{w}]$, have fully overlapping spans. We observe that for the additional matrix

$$\mathbf{T}_3 = \text{diag}\left(\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}\right) \tag{3.14}$$

we have that $[\mathbf{V} \quad \mathbf{T}_3\mathbf{V}] = \mathbf{H}_8$, where $\mathbf{H}_8$ is the 8×8 Hadamard matrix. In the following, we generalize the above observations and show that Hadamard designs provide the conditions for perfect alignment and linear independence for our problem.

Let $m = 2$, $N = 2^L$, and $\mathbf{X}_i = \mathbf{I}_{2^{i-1}} \otimes \text{blkdiag}\left(\mathbf{I}_{\frac{N}{2^i}}, -\mathbf{I}_{\frac{N}{2^i}}\right)$, for $i \in [L]$, and

consider the set[6]

$$\mathcal{H}_N = \left\{ \left(\prod_{i=1}^{L} \mathbf{X}_i^{x_i} \right) \mathbf{w} : x_i \in \{0,1\} \right\}. \tag{3.15}$$

Then, we have the following key lemma.

Lemma 1. *Let an $N \times N$ Hadamard matrix of the Sylvester's construction [98]*

$$\mathbf{H}_N \stackrel{\triangle}{=} \begin{bmatrix} \mathbf{H}_{\frac{N}{2}} & \mathbf{H}_{\frac{N}{2}} \\ \mathbf{H}_{\frac{N}{2}} & -\mathbf{H}_{\frac{N}{2}} \end{bmatrix}, \tag{3.16}$$

with $\mathbf{H}_1 = 1$. Then, $\mathbf{H}_N$ is full-rank with mutually orthogonal columns, that are the N elements of $\mathcal{H}_N$.

Proof. The proof can be found in the Appendix. □

Example 1. *To illustrate the connection between the Hadamard matrix $\mathbf{H}_N$ and its dots-on-a-lattice representation $\mathcal{H}_N$ we "decompose" the Hadamard matrix of order 4*

$$\mathbf{H}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} = [\mathbf{w} \;\; \mathbf{X}_2\mathbf{w} \;\; \mathbf{X}_1\mathbf{w} \;\; \mathbf{X}_2\mathbf{X}_1\mathbf{w}], \tag{3.17}$$

where $\mathbf{X}_1 = diag\begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix}$, $\mathbf{X}_2 = diag\begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}$, and $\mathbf{w} = \mathbf{1}_{4\times 1}$. Due to the commutativity of $\mathbf{X}_1$ and $\mathbf{X}_2$, the columns of $\mathbf{H}_4$ are also the elements of $\mathcal{H}_4 = \{\mathbf{w}, \mathbf{X}_1\mathbf{w}, \mathbf{X}_2\mathbf{w}, \mathbf{X}_1\mathbf{X}_2\mathbf{w}\}$.

Now, we will construct a matrix $\mathbf{V}_i$, whose columns are generated using the product-structure of $\mathcal{H}_N$, with the only difference that we omit a single $\mathbf{X}_i$ matrix from its construction. That is, we generate $\mathbf{V}_i$ with columns in the set

$$\mathcal{V}_i = \left\{ \left(\prod_{s=1, s\neq i}^{L} \mathbf{X}_s^{x_s} \right) \mathbf{w} : x_s \in \{0,1\} \right\}. \tag{3.18}$$

The space of $\mathbf{V}_i$ is invariant with respect to all $\mathbf{X}_s$, $s \neq i$, since the corresponding lattice representation wraps around itself due to $\mathbf{X}_s^2 = \mathbf{I}_N$, that is

$$\mathcal{L}(\mathbf{V}_i) = \mathcal{L}(\mathbf{X}_s\mathbf{V}_i),\ \forall s \neq i \Leftrightarrow \text{colspan}(\mathbf{V}_i) = \text{colspan}(\mathbf{X}_s\mathbf{V}_i),\ \forall s \neq i. \tag{3.19}$$

[6] In our general code constructions, we set L to be k.

Additionally, we have

$$\mathcal{L}(\mathbf{X}_i\mathbf{V}_i) = \left\{ \mathbf{e}_i + \sum_{s=1, s\neq i}^{L} x_s \mathbf{e}_s : x_s \in \{0,1\} \right\},$$

and we observe that $\mathcal{L}(\mathbf{X}_i\mathbf{V}_i) \cap \mathcal{L}(\mathbf{V}_i) = \emptyset$, i.e., $\mathcal{L}(\mathbf{V}_i)$ does not include any points with nonzero x_i coordinates. Then, due to the orthogonality of elements within $\mathcal{H}_N$, we have

$$|\mathcal{L}(\mathbf{V}_i)| = |\mathcal{L}(\mathbf{X}_s\mathbf{V}_i)| = \text{rank}(\mathbf{V}_i) = \text{rank}(\mathbf{X}_i\mathbf{V}_i) = N/2,$$

for any $i \neq s$. Hence, we obtain the following lemma for the set $\mathcal{H}_N$ and its associated $\mathcal{L}$ map.

Lemma 2. *For any $i, j \in [L]$ we have that*

$$\textit{rank}([\mathbf{V}_i \quad \mathbf{X}_j\mathbf{V}_i]) = |\mathcal{L}(\mathbf{V}_i) \cup \mathcal{L}(\mathbf{X}_j\mathbf{V}_i)| = \begin{cases} N, & i = j, \\ \frac{N}{2}, & i \neq j. \end{cases}$$

In Fig. 3.4, we give an illustrative example of the aforementioned definitions and properties. For $N = 2^3$, we consider $\mathbf{H}_8$ and $\mathbf{V}_3$ along with the matrix product $\mathbf{X}_2\mathbf{V}_3$ and their corresponding lattice representations.

To conclude this section, we have showed that starting from Hadamard matrices, we could obtain $\mathbf{X}_i$ and $\mathbf{V}_i$ matrices that have the perfect alignment properties of Eq. (3.5) required by our repair optimization problem. We will use these $\mathbf{X}_i$ matrices to build the coding matrices of our repair optimal code.

3.4 A Repair Optimal 2-Parity MDS Code

Code Construction 1. *Let the file size be $M = k2^{k+1}$ and let a $(k+2, k)$-MDS storage code with coding matrices*

$$\mathbf{A}_i = a_i\mathbf{X}_i + b_i\mathbf{X}_{k+1} + \mathbf{I}_N, \; i \in [k], \tag{3.20}$$

where $N = 2^{k+1}$,

$$\mathbf{X}_i = \mathbf{I}_{2^{i-1}} \otimes \textit{blkdiag}\left(\mathbf{I}_{\frac{N}{2^i}}, -\mathbf{I}_{\frac{N}{2^i}}\right), \tag{3.21}$$

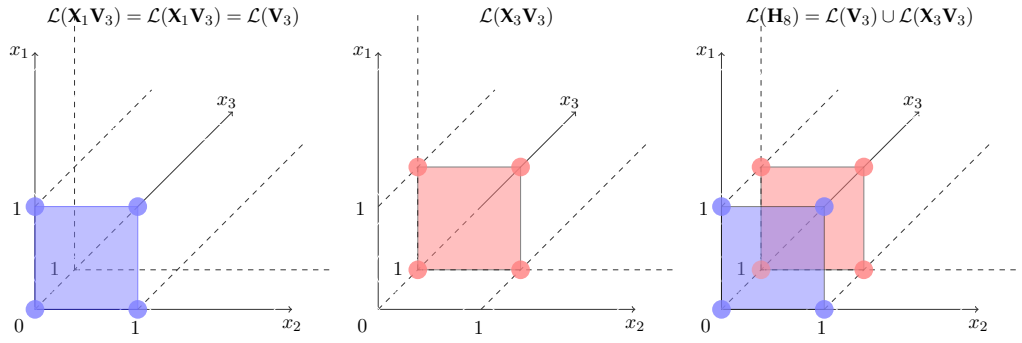


Figure 3.4: We set $N = 8$ and show the dots representation of $\mathbf{V}_3$, $\mathbf{X}_1\mathbf{V}_3$, $\mathbf{X}_2\mathbf{V}_3$, $\mathbf{X}_3\mathbf{V}_3$, and $\mathbf{H}_8$. Observe that we have perfect overlap for the cases of the $\mathbf{V}_3$, $\mathbf{X}_1\mathbf{V}_3$, and $\mathbf{X}_2\mathbf{V}_3$ matrices, which however do not intersect with $\mathbf{X}_3\mathbf{V}_3$. Hence, a union of the two distinct lattice point sets produces the lattice points of $\mathcal{L}(\mathbf{H}_8)$.

and a_i, b_i satisfy $a_i^2 - b_i^2 = -1$, for all $i \in [k]$.[7]

Theorem 1. *There exists a finite field $\mathbb{F}_q$ of prime and odd characteristic and order $q \geq 2k+3$ and explicit, non-zero constants $a_i, b_i \in \mathbb{F}_q$, $\forall i \in [k]$, such that the $(k+2, k)$ storage code in Eq. (3.20) is a repair optimal MDS storage code.*

In Fig. 3, we give the coding matrices of a $(5, 3)$-MDS code over $\mathbb{F}_{11}$ based on our construction. We would like to note that the field over which we construct our codes needs to have prime characteristic that is not equal to 2. This requirement is posed due to the fact that in a field whose characteristic is 2, the element -1 be equal to 1 and all our coding matrices will be equal to identity.

Remark 3. *The code constructions presented here have generator matrices that are as sparse as possible over $\mathbb{F}_q$, since any additional sparsity would violate the MDS property. This creates the additional benefit of minimum update complexity (at least over $\mathbb{F}_q$), when elements of the stored data object change.*

[7] We use -1 to denote the field element $q-1$ in $\mathbb{F}_q$.

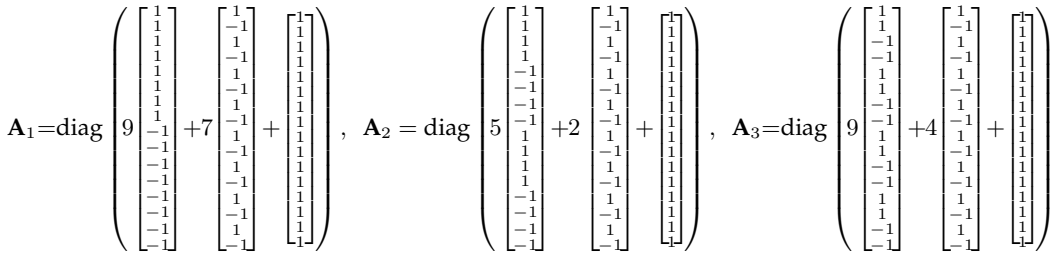


Figure 3.5: The coding matrices of a repair optimal $(5, 3)$-MDS code over $\mathbb{F}_{11}$

3.5 Optimal Systematic Node Repair

In this section, we show that our code in Eq. (3.20) has optimal systematic node repairs.

Let systematic node $i = 1$ of the code in Eq. (3.20) fail. We will construct $\mathbf{V}_1$ such that it is *a common invariant subspace* of $\mathbf{X}_2, \mathbf{X}_3, \ldots, \mathbf{X}_{k+1}$, i.e., if

$$\text{colspan}(\mathbf{V}_1) = \text{colspan}(\mathbf{X}_2\mathbf{V}_1) = \ldots = \text{colspan}(\mathbf{X}_{k+1}\mathbf{V}_1), \tag{3.22}$$

then, $\mathbf{V}_1$ would completely overlap with $\mathbf{A}_s\mathbf{V}_1$, for $s \neq 1$, as desired. To see this, note that $\mathbf{A}_s\mathbf{V}_1 = a_s\mathbf{X}_s\mathbf{V}_1 + b_s\mathbf{X}_{k+1}\mathbf{V}_1 + \mathbf{V}_1$. Therefore, every column vector of $\mathbf{A}_s\mathbf{V}_1, s \neq 1$ is the sum of three column vectors: one from the span of $\mathbf{X}_s\mathbf{V}_1$, one from the span of $\mathbf{X}_{k+1}\mathbf{V}_1$ and one from $\mathbf{V}_1$ itself. If $\mathbf{V}_1$ is invariant to both $\mathbf{X}_s$ and $\mathbf{X}_{k+1}$, then every column vector of $\mathbf{A}_s\mathbf{V}_1$ is a sum of three column vectors from the span of $\mathbf{V}_1$ and therefore lies in the span of $\mathbf{V}_1$. Hence, if we satisfy Eq. (3.22), $\mathbf{A}_s\mathbf{V}_1$ will lie in the span of $\mathbf{V}_1$.

Consider now the repair of systematic node $i \in [k]$ of the code in Eq. (3.20). The coding matrix $\mathbf{A}_i$ corresponding to the lost systematic piece $\mathbf{f}_i$, holds one matrix, that is, $\mathbf{X}_i$, which is unique among all other coding matrices, $\mathbf{A}_s$, $s \in [k]\backslash i$. We pick the columns of the repair matrix as a set of $\frac{N}{2}$ vectors whose lattice representation is invariant to all $\mathbf{X}_s$ matrices but to one key matrix: the unique $\mathbf{X}_i$ component of $\mathbf{A}_i$. We construct the $N \times \frac{N}{2}$ repair matrix $\mathbf{V}_i$ whose columns are the elements of the set

$$\mathcal{V}_i = \left\{ \left(\prod_{s=1, s\neq i}^{k+1} \mathbf{X}_s^{x_s} \right) \mathbf{w} : x_s \in \{0, 1\} \right\}. \tag{3.23}$$

This repair matrix is used to multiply both the contents of parity node 1 and 2, that is, $\mathbf{V}_i^{(1)} = \mathbf{V}_i^{(2)} = \mathbf{V}_i$. During the repair, the useful (desired signal) space populated by $\mathbf{f}_i$ is

$$[\mathbf{V}_i \;\; \mathbf{A}_i\mathbf{V}_i] \tag{3.24}$$

and the interference space due to file part $\mathbf{f}_s$, $s \neq i$, is

$$[\mathbf{V}_i \;\; \mathbf{A}_s\mathbf{V}_i]\,. \tag{3.25}$$

Remember that an optimal solution to $\mathcal{R}_i$ requires the useful space to have rank N and each of the interference spaces rank $\frac{N}{2}$. Observe that the following holds for each of the interference spaces

$$\begin{aligned}\frac{N}{2} &\leq \text{rank}\left([\mathbf{V}_i \;\; (a_s\mathbf{X}_s + b_s\mathbf{X}_{k+1} + \mathbf{I}_N)\,\mathbf{V}_i]\right)\\ &\leq |\mathcal{L}\left(\mathbf{V}_i\right) \cup \mathcal{L}\left(\mathbf{X}_s\mathbf{V}_i\right) \cup \mathcal{L}\left(\mathbf{X}_{k+1}\mathbf{V}_i\right)| = |\mathcal{L}\left(\mathbf{V}_i\right)| = \frac{N}{2},\end{aligned} \tag{3.26}$$

for $s \neq i$, since

$$\mathcal{L}(\mathbf{X}_s\mathbf{V}_i) = \mathcal{L}(\mathbf{V}_i), s \in [k+1]\backslash i, \tag{3.27}$$

due to Lemma 2. Then, for the useful data space we have

$$\begin{aligned}N &\geq \text{rank}\left([\mathbf{V}_i \;\; \mathbf{A}_i\mathbf{V}_i]\right) = \text{rank}\left([\mathbf{V}_i \;\; (a_i\mathbf{X}_i + b_i\mathbf{X}_{k+1} + \mathbf{I}_N)\,\mathbf{V}_i]\right)\\ &\overset{(*)}{=} \text{rank}\left([\mathbf{V}_i \;\; \mathbf{X}_i\mathbf{V}_i]\right) = |\mathcal{L}\left(\mathbf{V}_i\right) \cup \mathcal{L}\left(\mathbf{X}_i\mathbf{V}_i\right)|\\ &= |\mathcal{L}\left(\mathbf{H}_N\right)| = N,\end{aligned} \tag{3.28}$$

for any $a_i \neq 0$, where $(*)$ comes from the fact that $(a_i\mathbf{X}_i + b_i\mathbf{X}_{k+1} + \mathbf{I}_N)\,\mathbf{V}_i$ is a linear combination of columns from $\mathbf{V}_i$, $\mathbf{X}_{k+1}\mathbf{V}_i$, and $\mathbf{X}_i\mathbf{V}_i$. However, the column spaces of $\mathbf{V}_i$ and $\mathbf{X}_{k+1}\mathbf{V}_i$ are identical, thus the column space of $(a_i\mathbf{X}_i + b_i\mathbf{X}_{k+1} + \mathbf{I}_N)\,\mathbf{V}_i$ can be generated by linear combinations of $\mathbf{X}_i\mathbf{V}_i$ and $\mathbf{V}_i$. Moreover, the matrix

$$[\mathbf{V}_i \;\; (a_i\mathbf{X}_i + b_i\mathbf{X}_{k+1} + \mathbf{I}_N)\,\mathbf{V}_i]$$

has $\mathbf{V}_i$ within its columns, thus its span is the same as the span of $[\mathbf{V}_i \;\; \mathbf{X}_i\mathbf{V}_i]$.

Therefore, by using $\mathbf{V}_i$ as a repair matrix, we are able to generate the minimum amount of interference and at the same time satisfy the full-rank constraint of $\mathcal{R}_i$. Hence, the repair matrix in Eq. (3.23) is an optimal solution for $\mathcal{R}_i$ and systematic node i can be optimally repaired by downloading $(k+1)\frac{N}{2}$ data equations, for all $i \in [k]$.

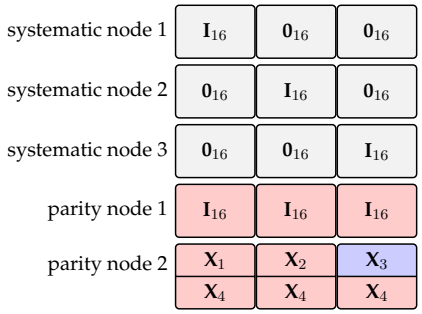
In Fig. 3.6, we present a more pictorial repair example where we consider the $(5, 3)$ case of our code in Eq. (3.20). We sketch the structure (or the generator matrix) of our code, where the a_i and b_i scalars and the $\mathbf{I}_{16}$ matrices in the $\mathbf{A}_i$ blocks are not mentioned for simplicity. In each $\mathbf{A}_i$ block of the second parity corresponds a unique "key" $\mathbf{X}_i$ matrix. During the repair of node 3, we use the repair matrix $\mathbf{V}_3$ that is defined in Eq. (3.23). Observe that matrices $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_4$ are used to construct $\mathbf{V}_3$. Hence, $\mathbf{V}_3$ is an invariant subspace of $\mathbf{X}_2$, $\mathbf{X}_3$, and $\mathbf{X}_4$. That way, interference aligns on the subspace $\mathbf{V}_3$, and the useful space $[\mathbf{V}_3\ \mathbf{X}_3\mathbf{V}_3]$ spans all N dimensions, since subspaces $\mathbf{V}_3$ and $\mathbf{X}_3\mathbf{V}_3$ are linearly independent. The alignment and full-rank properties are also exhibited, by the dots-on-a-lattice representation of the matrices.

3.6 Optimal Parity Repair

Performing optimal repair of parity nodes is conceptually more challenging than performing optimal repair of systematic nodes. This is because, as we will see in the following, repair space properties that hold during systematic node repairs, have to hold even after a change of basis. For the low-rate regime, $\frac{k}{n} \leq \frac{1}{2}$ repair optimal codes for both systematic and parity nodes have been discovered in [81,96]. However, the problem remained open for the high-rate regime. In particular, it is not known whether the systematic repair optimal codes of [17,18,99] admit optimal (or even non-trivially efficient) parity repair. We will explore this problem in this section. In particular, we will describe the additional properties to be satisfied by the coding matrices to ensure optimal parity repair (in addition to the properies that guarantee optimal systematic repair).

The code that we define in Eq. (3.20) admits optimal parity repair due to the fact that it satisfies all the space requirements that are analogous to the systematic repair setting. To see that we will rewrite our code in a new basis using a simple change of variables as the one in Fig. 3.7, where the parity nodes are transformed to "look like" systematic ones. This renaming of nodes provides the details under which parity repairs are understood in the systematic node repair framework.

As we will see in the following, the key ingredient of our construction that "unlocks" optimal repair for the first parity is the inclusion of the identity matrix in each $\mathbf{A}_i$. The same goes for the $\mathbf{X}_{k+1}$ matrix and the repair of the second parity.



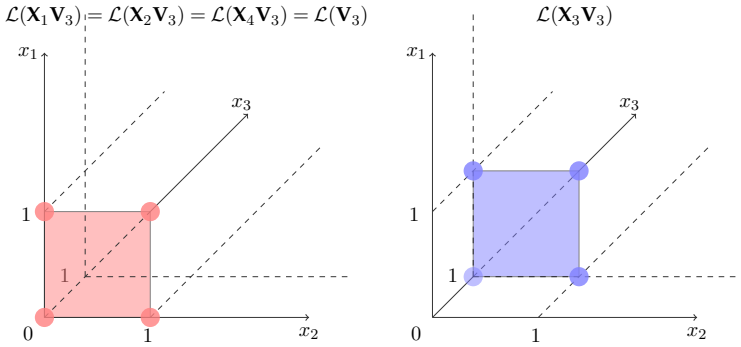


Figure 3.6: We illustrate some properties of the repair of node 3 in our $(6, 3)$ code. The red boxes in the coding matrix denote the matrices for which $\mathbf{V}_3$ is an invariant subspace. We also depict the dots-on-a-lattice representation of the repair spaces. The products of $\mathbf{V}_3$ with $\mathbf{X}^{x_1}, \mathbf{X}_2^{x_2}$ have the same lattice representation, whereas $\mathbf{X}_3^{x_3}\mathbf{V}_3$, $x_3 \neq 0$, correspond to a disjoint sets of lattice points

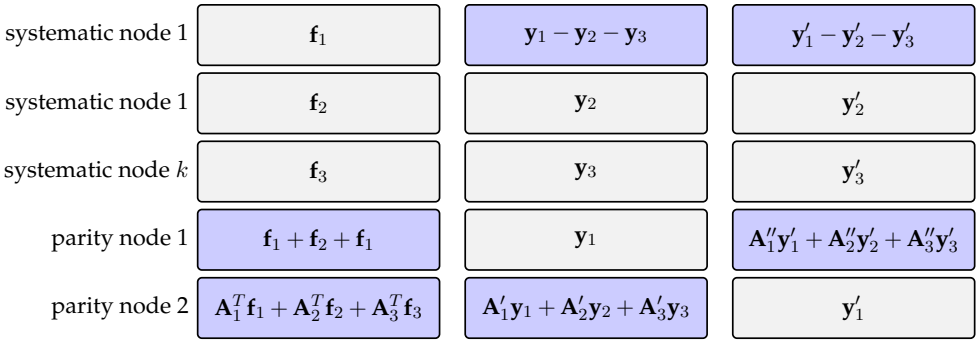


Figure 3.7: A change of variables for a $(5, 3)$-MDS code. We use it to represent a code in a way that we can treat the parity nodes as systematic nodes. These representations are equivalent with each other, in the sense that the code maintains its distance and repair properties.

Both these additionally included matrices refine the parity repair process such that optimality is feasible. Selecting appropriate constants a_i and b_i is also essential to our developments.

3.6.1 Repairing the first parity

Let the first parity node fail. We make a change of variables to obtain a new representation for our code in Eq. (3.20), where the first parity is a systematic node in an equivalent representation. We start with our $(k + 2, k)$-MDS storage code of Eq. (3.20)

$$\begin{bmatrix} \mathbf{I}_N & \mathbf{0}_N & \dots & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{I}_N & \dots & \mathbf{0}_N \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0}_N & \mathbf{0}_N & \dots & \mathbf{I}_N \\ \mathbf{I}_N & \mathbf{I}_N & \dots & \mathbf{I}_N \\ \mathbf{A}_1 & \mathbf{A}_2 & \dots & \mathbf{A}_k \end{bmatrix} \mathbf{f} \tag{3.29}$$

and make the following change of variables

$$\sum_{i=1}^{k} \mathbf{f}_i = \mathbf{y}_1, \tag{3.30}$$

$$\mathbf{f}_s = \mathbf{y}_s, \ s \in \{2, \dots, k\}. \tag{3.31}$$

We solve Eq. (3.30) and Eq. (3.31) for $\mathbf{f}_1$ in terms of the $\mathbf{y}_i$ variables and obtain

$$\mathbf{f}_1 = \mathbf{y}_1 - \sum_{s=2}^{k} \mathbf{y}_s. \tag{3.32}$$

Then, we plug (3.31) and (3.32) in Eq. (3.29), to have the equivalent representation

$$\begin{bmatrix} \mathbf{I}_N & -\mathbf{I}_N & \dots & -\mathbf{I}_N \\ \mathbf{0}_N & \mathbf{I}_N & \dots & \mathbf{0}_N \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0}_N & \mathbf{0}_N & \dots & \mathbf{I}_N \\ \mathbf{I}_N & \mathbf{0}_N & \dots & \mathbf{0}_N \\ \mathbf{A}_1 & \mathbf{A}_2-\mathbf{A}_1 & \dots & \mathbf{A}_k-\mathbf{A}_1. \end{bmatrix} \mathbf{y}, \tag{3.33}$$

where $\mathbf{y} = \left[\mathbf{y}_1^T \dots \mathbf{y}_k^T\right]^T \in \mathbb{F}_q^{kN}$. The first parity node of the code in Eq. (3.20) now corresponds to the node which contains $\mathbf{y}_1$ in the aforementioned representation. The coding matrices under this new representation are

$$\mathbf{A}_1 = a_1\mathbf{X}_1 + b_1\mathbf{X}_{k+1} + \mathbf{I}_N, \tag{3.34}$$

$$\mathbf{A}_s - \mathbf{A}_1 = a_s\mathbf{X}_s + (b_s - b_1)\mathbf{X}_{k+1} - a_1\mathbf{X}_1, \tag{3.35}$$

for $s \in \{2, \dots, k\}$. In contrast to the systematic node repair process, in the following we use a repair matrix of a slightly different structure. We construct the repair matrix $\mathbf{V}_a$ with columns in the set

$$\mathcal{V}_a = \left\{ \left(\prod_{s=2}^{k+1} \left(\mathbf{X}_1\mathbf{X}_s\right)^{x_s} \right) \mathbf{w} : x_s \in \{0,1\} \right\}. \tag{3.36}$$

Observe that this set is also a subset of $\mathcal{H}_N$. Then, to repair the node of (3.33) that contains $\mathbf{y}_1$ (i.e., the one that corresponds to the first parity node of (3.29)) we download $\mathbf{X}_1\mathbf{V}_a$ times the contents of the first parity in Eq. (3.33) and $\mathbf{V}_a$ times the contents of the second parity. Hence, during this repair, the useful space is spanned by

$$[\mathbf{X}_1\mathbf{V}_a \;\; \mathbf{A}_1\mathbf{V}_a] \tag{3.37}$$

and the interference space due to the "transformed file part" $\mathbf{y}_s$, $s \in \{2, \dots, k\}$, is

$$[\mathbf{X}_1\mathbf{V}_a \;\; (\mathbf{A}_s - \mathbf{A}_1)\mathbf{V}_a]. \tag{3.38}$$

Before we proceed, observe that the following rank conditions hold

$$\mathcal{L}(\mathbf{X}_1\mathbf{X}_s\mathbf{V}_a) = \mathcal{L}(\mathbf{V}_a) = \left\{ \left(\sum_{s=2}^{k+1} x_s \pmod 2 \right) \mathbf{e}_1 + \sum_{s=2}^{k+1} x_s \mathbf{e}_s;\ x_s \in \{0,1\} \right\} \tag{3.39}$$

$$\Leftrightarrow \mathcal{L}(\mathbf{X}_1\mathbf{V}_a) = \mathcal{L}(\mathbf{X}_s\mathbf{V}_a) = \left\{ \left(1 + \sum_{s=2}^{k+1} x_s \pmod 2 \right) \mathbf{e}_1 + \sum_{s=2}^{k+1} x_s \mathbf{e}_s;\ x_s \in \{0,1\} \right\} \tag{3.40}$$

$$\Rightarrow \mathcal{L}(\mathbf{X}_{s_1}\mathbf{V}_a) = \mathcal{L}(\mathbf{X}_{s_2}\mathbf{V}_a), \tag{3.41}$$

for any $s, s_1, s_2 \in [k+1]$. The above equations imply that

$$\mathcal{L}(\mathbf{V}_a) \cup \mathcal{L}(\mathbf{X}_1\mathbf{V}_a) = \left\{ \sum_{s=1}^{k+1} x_s \mathbf{e}_s;\ x_s \in \{0,1\} \right\} = \mathcal{L}(\mathbf{H}_N). \tag{3.42}$$

Therefore, we have the following for each of the interference spaces

$$\begin{aligned} \frac{N}{2} &\leq \text{rank}\left([\mathbf{X}_1\mathbf{V}_a \quad (a_s\mathbf{X}_s + (b_s - b_1)\mathbf{X}_{k+1} - a_1\mathbf{X}_1)\mathbf{V}_a]\right) \\ &\leq |\mathcal{L}(\mathbf{X}_1\mathbf{V}_a) \cup \mathcal{L}(\mathbf{X}_s\mathbf{V}_a) \cup \mathcal{L}(\mathbf{X}_{k+1}\mathbf{V}_a)| \\ &= |\mathcal{L}(\mathbf{X}_1\mathbf{V}_a)| = \frac{N}{2}. \end{aligned} \tag{3.43}$$

Moreover, for the useful data space we have

$$\begin{aligned} \text{rank}\left([\mathbf{X}_1\mathbf{V}_a \quad (a_1\mathbf{X}_1 + b_1\mathbf{X}_{k+1} + \mathbf{I}_N)\mathbf{V}_a]\right) &= \text{rank}([\mathbf{X}_1\mathbf{V}_a \ \ \mathbf{V}_a]) \\ &= |\mathcal{L}(\mathbf{V}_a) \cup \mathcal{L}(\mathbf{X}_1\mathbf{V}_a)| = |\mathcal{L}(\mathbf{H}_N)| = N, \end{aligned} \tag{3.44}$$

due to the same arguments as in Eq. (3.28). Thus, we can perform optimal repair of the node containing $\mathbf{y}_1$ in Eq. (3.33), which is equivalent to optimally repairing the first parity of our code in Eq. (3.20).

3.6.2 Repairing the second parity

Here, we have an additional step. We will first transform our coding matrices of (3.20) (via a symbol remapping) to an equivalent code, whose code matrix has a structure identical to the structure of the original code matrix. The difference is that in the new coding matrices, the last node looks like the first parity node of the original code structure, and the first parity node looks identical to the last node. This representation will allow us to prove the repair properties of the second

parity as we did for the first in Subsection 3.6.1. Without loss of generality, we can multiply any non-zero coding column block that multiplies the ith file part with a full-rank matrix $\mathbf{B}_i$ and maintain the same code and repair properties, as shown in [81]. We can rewrite the non-zero coded blocks in the following manner

$$\left[\begin{array}{c} \mathbf{I}_N \\ \mathbf{A}_i \end{array} \right] \mathbf{f}_i = \left[\begin{array}{c} \mathbf{I}_N \\ \mathbf{A}_i \end{array} \right] \mathbf{B}_i \mathbf{B}_i^{-1} \mathbf{f}_i = \left[\begin{array}{c} \mathbf{B}_i \\ \mathbf{A}_i \mathbf{B}_i \end{array} \right] \mathbf{f}_i' \tag{3.45}$$

where $\mathbf{B}_i$ is invertible and $\mathbf{f}_i' = \mathbf{B}_i^{-1} \mathbf{f}_i$. Then, If we can repair $\mathbf{f}_i'$, we can also repair $\mathbf{f}_i$ by multiplying $\mathbf{f}_i'$ with $\mathbf{B}_i$. This does not incur any additional repair bandwidth compared to the repair of $\mathbf{f}_i'$.

In the following derivations, we use the fact that $\mathbf{X}_s^2 = \mathbf{I}_N$, for any $s \in [k+1]$. We multiply the i-th block of (3.20) with $a_i \mathbf{X}_i - b_i \mathbf{X}_{k+1} + \mathbf{I}_N$ to obtain

$$\begin{aligned} \left[\begin{array}{c} \mathbf{I}_N \\ a_i \mathbf{X}_i + b_i \mathbf{X}_{k+1} + \mathbf{I}_N \end{array} \right] &\equiv \left[\begin{array}{c} a_i \mathbf{X}_i - b_i \mathbf{X}_{k+1} + \mathbf{I}_N \\ (a_i \mathbf{X}_i - b_i \mathbf{X}_{k+1} + \mathbf{I}_N)(a_i \mathbf{X}_i + b_i \mathbf{X}_{k+1} + \mathbf{I}_N) \end{array} \right] \\ &= \left[\begin{array}{c} a_i \mathbf{X}_i - b_i \mathbf{X}_{k+1} + \mathbf{I}_N \\ (a_i \mathbf{X}_i + \mathbf{I}_N)^2 - b_i^2 \mathbf{I}_N \end{array} \right] \\ &\equiv \left[\begin{array}{c} a_i \mathbf{X}_i - b_i \mathbf{X}_{k+1} + \mathbf{I}_N \\ 2a_i \mathbf{X}_i + (a_i^2 - b_i^2 + 1)\mathbf{I}_N. \end{array} \right] \overset{(*)}{=} \left[\begin{array}{c} a_i \mathbf{X}_i - b_i \mathbf{X}_{k+1} + \mathbf{I}_N \\ 2a_i \mathbf{X}_i \end{array} \right], \end{aligned} \tag{3.46}$$

where in $(*)$ we use the fact that $a_i^2 - b_i^2 + 1 = 0$. We continue by multiplying the i-th column block with $(a_i)^{-1} \mathbf{X}_i$ to obtain

$$\left[\begin{array}{c} a_i \mathbf{X}_i - b_i \mathbf{X}_{k+1} + \mathbf{I}_N \\ 2a_i \mathbf{X}_i \end{array} \right] \equiv \left[\begin{array}{c} \mathbf{I}_N - a_i^{-1} b_i \mathbf{X}_{k+1} \mathbf{X}_i + a_i^{-1} \mathbf{X}_i \\ 2\mathbf{I}_N \end{array} \right]. \tag{3.47}$$

Hence,

$$2\mathbf{A}_i^{-1} = \mathbf{I}_N - a_i^{-1} b_i \mathbf{X}_{k+1} \mathbf{X}_i + a_i^{-1} \mathbf{X}_i), \; i \in [k]. \tag{3.48}$$

Then, we rewrite our original code as

$$\left[\begin{array}{cccc} 2\mathbf{A}_1^{-1} & \mathbf{0}_N & \dots & \mathbf{0}_N \\ \mathbf{0}_N & 2\mathbf{A}_2^{-1} & \dots & \mathbf{0}_N \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0}_N & \mathbf{0}_N & \dots & 2\mathbf{A}_k^{-1} \\ 2\mathbf{A}_1^{-1} & 2\mathbf{A}_2^{-1} & \dots & 2\mathbf{A}_k^{-1} \\ 2\mathbf{I}_N & 2\mathbf{I}_N & \dots & 2\mathbf{I}_N \end{array} \right] \mathbf{f}', \tag{3.49}$$

where $\mathbf{f}'$ is a full row-rank transformation of $\mathbf{f}$. We will focus on the following code matrix

$$\begin{bmatrix} \mathbf{I}_N & \mathbf{0}_N & \dots & \mathbf{0}_N \\ \mathbf{0}_N & \mathbf{I}_N & \dots & \mathbf{0}_N \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0}_N & \mathbf{0}_N & \dots & \mathbf{I}_N \\ 2\mathbf{A}_i^{-1} & 2\mathbf{A}_i^{-1} & \dots & 2\mathbf{A}_i^{-1} \\ \mathbf{I}_N & \mathbf{I}_N & \dots & \mathbf{I}_N \end{bmatrix} \mathbf{f}'. \tag{3.50}$$

To repair our second parity, we can use the same repair matrices used for the repair of the second parity of the above construction, with a slight manupulation. We simply need to multiply the repair matrices corresponding to systematic parts with $2^{-1}\mathbf{A}_i^{-1}$.

We proceed in the same manner that we handled the first parity repair. We make a change of variables such that the second parity becomes a systematic node in a new representation

$$\sum_{i=1}^{k} \mathbf{f}_i' = \mathbf{y}_1' \tag{3.51}$$

and obtain the equivalent form

$$\begin{bmatrix} \mathbf{I}_N & -\mathbf{I}_N & \dots & -\mathbf{I}_N \\ \mathbf{0}_N & \mathbf{I}_N & \dots & \mathbf{0}_N \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0}_N & \mathbf{0}_N & \dots & \mathbf{I}_N \\ 2\mathbf{A}_1^{-1} & 2\mathbf{A}_2^{-1}-2\mathbf{A}_1^{-1} & \dots & 2\mathbf{A}_k^{-1}-2\mathbf{A}_1^{-1} \\ \mathbf{I}_N & \mathbf{0}_N & \dots & \mathbf{0}_N. \end{bmatrix} \mathbf{y}', \tag{3.52}$$

where

$$2\mathbf{A}_1^{-1} = \mathbf{I}_N - a_1^{-1}b_1\mathbf{X}_{k+1}\mathbf{X}_1 + a_1^{-1}\mathbf{X}_1, \tag{3.53}$$

$$2\mathbf{A}_s^{-1} - 2\mathbf{A}_1^{-1} = a_s^{-1}\mathbf{X}_s - a_s^{-1}b_s\mathbf{X}_{k+1}\mathbf{X}_s + a_1^{-1}b_1\mathbf{X}_{k+1}\mathbf{X}_1 - a_1^{-1}\mathbf{X}_1. \tag{3.54}$$

Then, the parity node which corresponds to systematic node 1 here, can be repaired by using $\mathbf{V}_b$ with columns in the set

$$\mathcal{V}_b = \left\{ \left(\mathbf{X}_{k+1}^{x_{k+1}} \prod_{s=1}^{k} (\mathbf{X}_1\mathbf{X}_s)^{x_s} \right) \mathbf{w} : x_{k+1}, x_s \in \{0,1\} \right\}. \tag{3.55}$$

Again, the following equations hold

$$\mathcal{L}(\mathbf{X}_{k+1}\mathbf{V}_b) = \mathcal{L}(\mathbf{V}_b) = \left\{ \left(\sum_{s=2}^{k} x_s \pmod 2 \right) \mathbf{e}_1 + \sum_{s=2}^{k+1} x_s \mathbf{e}_s;\ x_s \in \{0,1\} \right\}, \tag{3.56}$$

$$\mathcal{L}(\mathbf{X}_{s_1}\mathbf{V}_b) = \mathcal{L}(\mathbf{X}_{s_2}\mathbf{V}_b) = \left\{ \left(1 + \sum_{s=2}^{k} x_s \pmod 2 \right) \mathbf{e}_1 + \sum_{s=2}^{k+1} x_s \mathbf{e}_s;\ x_s \in \{0,1\} \right\}, \tag{3.57}$$

$$\text{and } \mathcal{L}(\mathbf{X}_{s_1}\mathbf{X}_{k+1}\mathbf{V}_b) = \mathcal{L}(\mathbf{X}_{s_1}\mathbf{V}_b), \tag{3.58}$$

for all $s_1, s_2 \in [k]$. Hence, we have for the interfence space generated by component $\mathbf{y}'_s$, $s \in \{2, \ldots, k\}$

$$\begin{aligned} \frac{N}{2} &\leq \text{rank}\left(\left[\mathbf{X_1}\mathbf{V}_b \ \ (\hat{\mathbf{A}}_s - \hat{\mathbf{A}}_1)\mathbf{V}_b \right) \right) \\ &\leq |\mathcal{L}(\mathbf{X}_s\mathbf{V}_b) \cup \mathcal{L}(\mathbf{X}_1\mathbf{V}_b) \cup \mathcal{L}(\mathbf{X}_{k+1}\mathbf{X}_1\mathbf{V}_b) \cup \mathcal{L}(\mathbf{X}_{k+1}\mathbf{X}_s\mathbf{V}_b)| \\ &= |\mathcal{L}(\mathbf{X}_1\mathbf{V}_b) \cup \mathcal{L}(\mathbf{X}_s\mathbf{V}_b)| = \frac{N}{2}. \end{aligned} \tag{3.59}$$

Moreover, the useful space is full-rank

$$\text{rank}\left(\left[\mathbf{X_1}\mathbf{V}_b \ \ \left(\mathbf{I}_N - a_1^{-1}b_1\mathbf{X}_{k+1}\mathbf{X}_1 + a_1^{-1}\mathbf{X}_1 \right) \mathbf{V}_b \right] \right) = \text{rank}\left([\mathbf{X_1}\mathbf{V}_b \ \ \mathbf{V}_b] \right) = N. \tag{3.60}$$

Thus, we can perform optimal repair of the second parity of the code in Eq. (3.20), with repair bandwidth $(k+1)\frac{N}{2}$.

3.7 The MDS Property

In this section we give explicit conditions on the a_i, b_i constants, for all $i \in [k]$, and the order of the finite field $\mathbb{F}_q$, for which the code in Eq. (3.20) is MDS. We discuss the MDS property using the notion of data collectors (DCs), in the same manner that it was used in [34]. A DC can be considered as an external user that can connect and has complete access to the contents of some subset of k nodes. A storage code where each node expends $\frac{M}{k}$ worth of storage, has the MDS property when all possible $\binom{n}{k}$ DCs can decode the file $\mathbf{f}$. We can show that testing the MDS property is equivalent to checking the rank of a specific matrix associated with each DC. This DC matrix is the vertical concatenation of the k stacks of equations stored by the nodes that the DC connects to. If all $\binom{n}{k}$ DC matrices are full-rank, then we declare that the storage code has the MDS property.

We start with a DC that connects to systematic nodes $\{1 \ldots, k-1\}$ and the

first parity node. The determinant of the corresponding DC matrix is

$$\det\left(\left[\begin{array}{ccc|c} \mathbf{I}_N & \dots & \mathbf{0}_{N\times N} & \mathbf{0}_{N\times N} \\ \vdots & & \vdots & \vdots \\ \mathbf{0}_{N\times N} & \dots & \mathbf{I}_N & \mathbf{0}_{N\times N} \\ \hline \mathbf{I}_N & \dots & \mathbf{I}_N & \mathbf{I}_N \end{array}\right]\right) = \det\left(\mathbf{I}_N\right) \neq 0, \tag{3.61}$$

since $\mathbf{I}_N$ is a full-rank diagonal matrix. We continue by considering a DC that connects to systematic nodes $\{1\dots,k-1\}$ and the second parity node. For that we have

$$\det\left(\left[\begin{array}{ccc|c} \mathbf{I}_N & \dots & \mathbf{0}_{N\times N} & \mathbf{0}_{N\times N} \\ \vdots & & \vdots & \vdots \\ \mathbf{0}_{N\times N} & \dots & \mathbf{I}_N & \mathbf{0}_{N\times N} \\ \hline \mathbf{A}_1 & \dots & \mathbf{A}_{k-1} & \mathbf{A}_k \end{array}\right]\right) = \det\left(\mathbf{A}_k\right) \neq 0, \tag{3.62}$$

due to $\mathbf{A}_k$ being full-rank.

Finally, we consider DCs that connect to $k-2$ systematic nodes and both parity nodes. Let a DC that connects to systematic node

$$\{1\dots,k-2\}$$

and the two parities. The corresponding DC matrix is

$$\left[\begin{array}{ccc|cc} \mathbf{I}_N & \dots & \mathbf{0}_{N\times N} & \mathbf{0}_{N\times N} & \mathbf{0}_{N\times N} \\ \vdots & & \vdots & \vdots & \\ \mathbf{0}_{N\times N} & \dots & \mathbf{I}_N & \mathbf{0}_{N\times N} & \mathbf{0}_{N\times N} \\ \hline \mathbf{I}_N & \dots & \mathbf{I}_N & \mathbf{I}_N & \mathbf{I}_N \\ \mathbf{A}_1 & \dots & \mathbf{A}_{k-2} & \mathbf{A}_{k-1} & \mathbf{A}_k \end{array}\right]. \tag{3.63}$$

The leftmost $(k-2)N$ columns of the matrix in Eq. (3.63) are linearly independent, due to the upper-left identity block. Moreover, the leftmost $(k-2)N$ columns are linearly independent with the rightmost $2N$, using an analogous argument, if and only if that the right most $2N$ columns are linearly independent. Hence, we need to only check the rank of the sub-matrix

$$\left[\begin{array}{cc} \mathbf{I}_N & \mathbf{I}_N \\ \mathbf{A}_{k-1} & \mathbf{A}_k \end{array}\right]. \tag{3.64}$$

In the general case, a DC that connects to some $k-2$ subset of systematic nodes and the two parities has a corresponding matrix where the following block needs to be full-rank so that the MDS property can be satisfied

$$\left[\begin{array}{cc} \mathbf{I}_N & \mathbf{I}_N \\ \mathbf{A}_i & \mathbf{A}_j \end{array}\right], \tag{3.65}$$

for $i, j \in [k]$ and $i \neq j$. The code is MDS when for all $i, j \in [k]$

$$\begin{aligned}
&\text{rank}\left(\begin{bmatrix} \mathbf{I}_N & \mathbf{I}_N \\ a_i\mathbf{X}_i + b_i\mathbf{X}_{k+1} + \mathbf{I}_N & a_j\mathbf{X}_j + b_j\mathbf{X}_{k+1} + \mathbf{I}_N \end{bmatrix}\right) \\
&= \text{rank}\left(\begin{bmatrix} \mathbf{I}_N & \mathbf{I}_N \\ a_i\mathbf{X}_i + b_i\mathbf{X}_{k+1} + \mathbf{I}_N & a_j\mathbf{X}_j + b_j\mathbf{X}_{k+1} + \mathbf{I}_N \end{bmatrix} \times \begin{bmatrix} \mathbf{I}_N & \mathbf{I}_N \\ \mathbf{0}_{N\times N} & -\mathbf{I}_N \end{bmatrix}\right) \\
&= \text{rank}\left(\begin{bmatrix} \mathbf{I}_N & 0 \\ a_i\mathbf{X}_i + b_i\mathbf{X}_{k+1} + \mathbf{I}_N & a_i\mathbf{X}_i - a_j\mathbf{X}_j + (b_i - b_j)\mathbf{X}_{k+1} \end{bmatrix}\right) \\
&= N + \text{rank}\,(a_i\mathbf{X}_i - a_j\mathbf{X}_j + (b_i - b_j)\mathbf{X}_{k+1}) = 2N, \qquad (3.66)
\end{aligned}$$

which is true if

$$\text{rank}\,(a_i\mathbf{X}_i - a_j\mathbf{X}_j + (b_i - b_j)\mathbf{X}_{k+1}) = N. \qquad (3.67)$$

Since the diagonal elements of $\mathbf{X}_i$ are $\{\pm 1\}$, the previous requirement yields the following lemma.

Lemma 3. *The code in Eq. (3.20) is MDS when*

$$i)\ \ a_i - a_j + (b_i - b_j) \neq 0, \qquad (3.68)$$

$$ii)\ \ a_i + a_j - (b_i - b_j) \neq 0, \qquad (3.69)$$

$$iii)\ \ a_i - a_j - (b_i - b_j) \neq 0, \qquad (3.70)$$

$$\textit{and } iv)\ \ a_i + a_j + (b_i - b_j) \neq 0, \qquad (3.71)$$

for all $i \neq j \in [k]$.

Now, remember that our initial constraint on the a_i and b_i constants was

$$a_i^2 - b_i^2 = -1 \Leftrightarrow (a_i - b_i)(a_i + b_i) = -1. \qquad (3.72)$$

One solution to the previous equation is the following

$$a_i - b_i = x_i, \qquad (3.73)$$

$$a_i + b_i = -x_i^{-1}. \qquad (3.74)$$

If we input the above solution to (3.72), then the MDS equations (3.71) become

$$\begin{aligned} a_i - a_j + (b_i - b_j) &= a_i + b_i - (a_i + b_j) \\ &= -x_i^{-1} + x_j^{-1} \neq 0 \\ &\Leftrightarrow x_i^{-1} \neq x_j^{-1}, \end{aligned} \tag{3.75}$$

$$\begin{aligned} a_i + a_j - (b_i - b_j) &= a_i - b_i + a_j + b_j \\ &= x_i - x_j^{-1} \neq 0 \\ &\Leftrightarrow x_i \neq x_j^{-1}, \end{aligned} \tag{3.76}$$

$$\begin{aligned} a_i - a_j - (b_i - b_j) &= a_i - b_i - (a_j - b_j) \\ &= x_i - x_j \neq 0 \\ &\Leftrightarrow x_i \neq x_j, \end{aligned} \tag{3.77}$$

$$\begin{aligned} a_i + a_j + (b_i - b_j) &= a_i + b_i + a_j - b_j \\ &= -x_i^{-1} + x_j \neq 0 \\ &\Leftrightarrow x_i^{-1} \neq x_j. \end{aligned} \tag{3.78}$$

The above conditions can be equivalently stated as

$$x_i \neq x_j \text{ and } x_i x_j \neq 1, \tag{3.79}$$

for any $i \neq j \in [k]$.

Then, consider a field $\mathbb{F}_q$ of size q. The set of x_is that satisfies our MDS requirements, is such in which no two elements are inverses of each other. Note that the non-zero elements of a finite field can be partitioned into two equal sized partitions, where the multiplicative inverse of an element from the first partition lies in the second partition (and vice-versa). If we additionally do not consider $x_i \in \{1, q-1\}$, then we are left with $\frac{q-3}{2}$ elements. Therefore, we can consider a field of prime characteristic p, such as its order q has the property

$$k \leq \frac{q-3}{2} \Leftrightarrow q \geq 2k + 3 \tag{3.80}$$

and obtain $x_1, \ldots, x_k$ such that our requirements are satisfied. Then, the elements a_i and b_i, for all $i \in [k]$, can be obtained through the following equations

$$a_i = 2^{-1} x_i - 2^{-1} x_i^{-1}, \tag{3.81}$$

$$b_i = -2^{-1} x_i - 2^{-1} x_i^{-1}. \tag{3.82}$$

Observe that the above solutions yield $a_i \neq 0$ (that is needed for successful repair), for all $i \in [k]$, when $x_i \notin \{0, 1, q-1\}$. Therefore a prime characteristic field of order greater than, or equal to $2k+3$ always suffices to obtain the MDS property.

3.8 Generalizing to more than 2 parities

3.8.1 m-parity codes with optimal systematic repair

We generalize the Hadamard design construction of Section 3.4 and of the code in [78], to construct $(k+m, k)$-MDS storage codes for file sizes $M = km^k$. Our constructions are based on a generalization of the Sylvester construction for complex Hadamard matrices that use m^{th} roots of unity. We generate these matrices as

$$\mathbf{H}_{m^k} = \mathbf{H}_m \otimes \mathbf{H}_{m^{k-1}}, \tag{3.83}$$

where $\mathbf{H}_m$ is the m-point Discrete Fourier Transform matrix over a finite field. For example, for $m = 3$ and $\mathbb{F}_7$, we have

$$\mathbf{H}_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \rho & \rho^2 \\ 1 & \rho^2 & \rho \end{bmatrix} \text{ and } \mathbf{H}_9 = \begin{bmatrix} \mathbf{H}_3 & \mathbf{H}_3 & \mathbf{H}_3 \\ \mathbf{H}_3 & \rho\mathbf{H}_3 & \rho^2\mathbf{H}_3 \\ \mathbf{H}_3 & \rho^2\mathbf{H}_3 & \rho\mathbf{H}_3 \end{bmatrix}, \tag{3.84}$$

where $\rho = 2$. Then, we consider the set

$$\mathcal{H}_{m^k} = \left\{ \left(\prod_{i=1}^{k} \mathbf{X}_i^{x_i} \right) \mathbf{w} : x_i \in \{0, 1, \ldots, m-1\} \right\}, \tag{3.85}$$

where $\mathbf{w} = \mathbf{1}_{m^k \times 1}$ and

$$\mathbf{X}_i = \mathbf{I}_{m^{i-1}} \otimes \text{blkdiag}\left(\mathbf{I}_{\frac{N}{m^i}}, \rho \mathbf{I}_{\frac{N}{m^i}}, \ldots, \rho^{m-1} \mathbf{I}_{\frac{N}{m^i}} \right). \tag{3.86}$$

Here, ρ denotes an m^{th} root of unity which yields

$$\mathbf{X}_i^m = \mathbf{I}_{m^k}. \tag{3.87}$$

As with the $m = 2$ case, there is a one-to-one correspondence between the elements of the set $\mathcal{H}_{m^k}$ and the columns of $\mathbf{H}_{m^k}$. The general m proof for that property follows the same manner of the $m = 2$ case, thus we omit it.

Remark 4. *Observe that for the full-rank property of the spaces in the $n - k = 2$ code construction we required $\mathbf{H}_N^T \mathbf{H}_N = \mathbf{I}_N$, for $N = 2^{k+1}$, i.e., that the useful spaces have orthogonal columns. However, linear independece suffices for our purposes. In our m-parity code construction, we only require that $\mathbf{H}_{m^k}$ is full-rank.*

We proceed by ebstablishing that there exist fields in which $\mathbf{H}_{m^k}$ is full-rank. First observe that

$$\mathbf{H}_{m^k} = \underbrace{\mathbf{H}_m \otimes \ldots \otimes \mathbf{H}_m}_{k \text{ times}}. \tag{3.88}$$

Hence,

$$|\mathbf{H}_{m^k}| = \prod_{i=1}^{k} |\mathbf{H}_m|^m. \tag{3.89}$$

Therefore, $\mathbf{H}_{m^k}$ is full-rank, if and only if $|\mathbf{H}_m| \neq 0$. Observe, that since $\mathbf{H}_m$ is the m-point DFT matrix, then it follows the Vandermonde structure, and its determinant is given by

$$|\mathbf{H}_m| = \prod_{1 \leq i < j \leq m} (\alpha_i - \alpha_j), \tag{3.90}$$

with $\alpha_i = \rho^{i-1}$ and $i = 1, \ldots, m$. Hence, the full-rank property is obtained when $\rho^i \neq \rho^j$ for all $i \neq j \in \{0, \ldots, q-1\}$. Therefore, to maintain the full-rank property of $\mathbf{H}_{m^k}$, the finite field over which we operate should be chosen such that all m^{th} roots of unity are distinct. The number of distinct m^{th} roots of unity over a finite field $\mathbb{F}_q$ is given by the number of (distinct) solutions of the equation $x^m = 1$. This is equal to the order of the cyclic group that generates m^{th} roots of unity within the multiplicative group of $\mathbb{F}_q$. This subgroup has order m, i.e., our $\mathbf{H}_m$ matrix is full-rank, when m divides $q-1$ [65], i.e., when $q = C \cdot m + 1$, for some non-negative C.

Code Construction 2. *Our $(k+m, k)$-MDS code encodes a file $\mathbf{f}$ of size $M = km^k$ in the manner of*

$$\begin{bmatrix} \mathbf{I}_{km^k} \\ \mathbf{A}^{(k,m)} \end{bmatrix} \mathbf{f}, \tag{3.91}$$

where

$$\mathbf{A}^{(k,m)} = \begin{bmatrix} \mathbf{I}_{m^k} & \mathbf{I}_{m^k} & \ldots & \mathbf{I}_{m^k} \\ \lambda_{1,1}\mathbf{X}_1 & \lambda_{1,2}\mathbf{X}_2 & \ldots & \lambda_{1,k}\mathbf{X}_k \\ \lambda_{2,1}\mathbf{X}_1^2 & \lambda_{2,2}\mathbf{X}_2^2 & \ldots & \lambda_{2,k}\mathbf{X}_k^2 \\ \vdots & & & \\ \lambda_{m-1,k}\mathbf{X}_1^{m-1} & \lambda_{m-1,2}\mathbf{X}_2^{m-1} & \ldots & \lambda_{m-1,k}\mathbf{X}_k^{m-1} \end{bmatrix}, \tag{3.92}$$

with $\lambda_{i,j} \in \mathbb{F}_q$.

3.8.1.1 Optimal repair of the systematic nodes

For this code, let systematic node $i \in [k]$ fail. Then, to repair it we construct the repair matrix $\mathbf{V}_i$ that has as columns the elements of the set

$$\mathcal{V}_i = \left\{ \left(\prod_{s=1, s\neq i}^{k} \mathbf{X}_s^{x_s} \right) \mathbf{w} : x_s \in \{0, 1, \ldots, m-1\} \right\}. \tag{3.93}$$

This matrix is used to multiply the contents of each of the parity nodes. Here, the useful space during the repair is given by

$$\left[\mathbf{V}_i \;\; \mathbf{X}_i \mathbf{V}_i \;\; \mathbf{X}_i^2 \mathbf{V}_i \; \ldots \; \mathbf{X}_i^{m-1} \mathbf{V}_i \right] \tag{3.94}$$

and the interference space generated by systematic component $j \neq i$ is spanned by

$$\left[\mathbf{V}_i \;\; \mathbf{X}_j \mathbf{V}_i \;\; \mathbf{X}_j^2 \mathbf{V}_i \; \ldots \; \mathbf{X}_j^{m-1} \mathbf{V}_i \right]. \tag{3.95}$$

Due to the modulus-m property of the powers of the $\mathbf{X}_i$ matrices, we obtain the following under the lattice representation

$$\mathcal{L}\left(\mathbf{V}_i\right) = \mathcal{L}\left(\mathbf{X}_j^l \mathbf{V}_i\right) \text{ and } \mathcal{L}\left(\mathbf{X}_i^{l_1} \mathbf{V}_i\right) \cap \mathcal{L}\left(\mathbf{X}_i^{l_2} \mathbf{V}_i\right) = \emptyset, \tag{3.96}$$

for any $j \in [k] \neq i$, and $l, l_1, l_2 \in \{0, \ldots, m-1\}$, with $l_1 \neq l_2$. The above property and the fact that the elements of $\mathcal{H}_{m^k}$ are linearly independent lead us to the following lemma.

Lemma 4. *For any $i, j \in [k]$ we have that*

$$\begin{aligned} &\textit{rank}\left(\left[\mathbf{V}_i \;\; \mathbf{X}_j \mathbf{V}_i \;\; \mathbf{X}_j^2 \mathbf{V}_i \; \ldots \; \mathbf{X}_j^{m-1} \mathbf{V}_i \right] \right) \\ &= \left| \mathcal{L}(\mathbf{V}_i) \cup \mathcal{L}\left(\mathbf{X}_j \mathbf{V}_i\right) \cup \mathcal{L}\left(\mathbf{X}_j^2 \mathbf{V}_i\right) \cup \ldots \cup \mathcal{L}\left(\mathbf{X}_j^{m-1} \mathbf{V}_i\right) \right| \\ &= \begin{cases} m^k, & i = j, \\ m^{k-1}, & i \neq j. \end{cases} \end{aligned} \tag{3.97}$$

By Lemma 4 we see that each of the $k-1$ interference terms is confined within m^{k-1} dimensions and the full-rank property of the useful space is maintained. This is equivalent to stating that we can repair a single systematic node

failure by downloading exactly $m^k + (k-1)m^{k-1} = (n-1)m^{k-1}$ equations, which matches exactly the optimal repair bandwidth of [34].

In Fig. 3.8, we give a sketch of the generator matrix of a $(6,3)$ systematic-repair optimal code. While optimally repairable $(6,3)$ codes have been discovered previously [20,35,81,82,89,96], we use these parameters merely for exposition. Each parity block is associated with a specific key matrix $\mathbf{X}_i$. This, when considering the repair of node 3, allows a selection of $\mathbf{V}_3$ that is an invariant subspace to all matrices but to the key ones $\mathbf{X}_3$ and $\mathbf{X}_3^2$ which multiply the desired and lost file piece. This $\mathbf{V}_3$ enables perfect alignment of interference in m^2 dimensions, while ensuring a full-rank m^3 useful space.

3.8.1.2 The MDS property

We establish the MDS property of our m-parity codes in a probabilistic way: we show that when we select the $\lambda_{i,j}$ variables uniformly at random over a sufficiently large finite field of prime characteristic and order $q = mC + 1$, then the code is MDS with probability arbitrarily close to 1. This is shown using the Schwartz-Zippel lemma [47,75] on a nonzero polynomial on $\lambda_{i,j}$s induced by the products of all possible DC matrix determinants.

Let a DC of the code in Eq. (3.91) that connects to $k-P$ systematic nodes and P parities. For simplicity consider that this is the DC that is connected to the last $k-P$ systematic nodes and the first P parity nodes. The induced determinant of the corresponding DC matrix will be zero if the following determinant is zero

$$\det\left(\begin{bmatrix} \mathbf{I}_{m^k} & \mathbf{I}_{m^k} & \dots & \mathbf{I}_{m^k} \\ \lambda_{1,1}\mathbf{X}_1 & \lambda_{1,2}\mathbf{X}_2 & \dots & \lambda_{1,k}\mathbf{X}_k \\ \lambda_{2,1}\mathbf{X}_1^2 & \lambda_{2,2}\mathbf{X}_2^2 & \dots & \lambda_{2,k}\mathbf{X}_k^2 \\ \vdots & & & \\ \lambda_{P-1,1}\mathbf{X}_1^{P-1} & \lambda_{P-1,2}\mathbf{X}_2^{P-1} & \dots & \lambda_{P-1,k}\mathbf{X}_k^{P-1} \\ \hline \mathbf{0}_{(k-P)m^k \times Pm^k} & & \mathbf{I}_{(k-P)m^k} & \end{bmatrix}\right) \tag{3.98}$$

$$= |\mathbf{I}_{(k-P)m^k}| \det\left(\begin{bmatrix} \mathbf{I}_{m^k} & \mathbf{I}_{m^k} & \dots & \mathbf{I}_{m^k} \\ \lambda_{1,1}\mathbf{X}_1 & \lambda_{1,2}\mathbf{X}_2 & \dots & \lambda_{1,P}\mathbf{X}_P \\ \lambda_{2,1}\mathbf{X}_1^2 & \lambda_{2,2}\mathbf{X}_2^2 & \dots & \lambda_{2,P}\mathbf{X}_P^2 \\ \vdots & & & \\ \lambda_{P-1,1}\mathbf{X}_1^{P-1} & \lambda_{P-1,2}\mathbf{X}_2^{P-1} & \dots & {}_{P-1,P}\mathbf{X}_P^{P-1} \end{bmatrix}\right). \tag{3.99}$$

Since each of the $\mathbf{X}_i$ matrices is diagonal, each column of the matrix in the right hand side of (3.99) has exactly P nonzero elements. These Pm^k columns can be

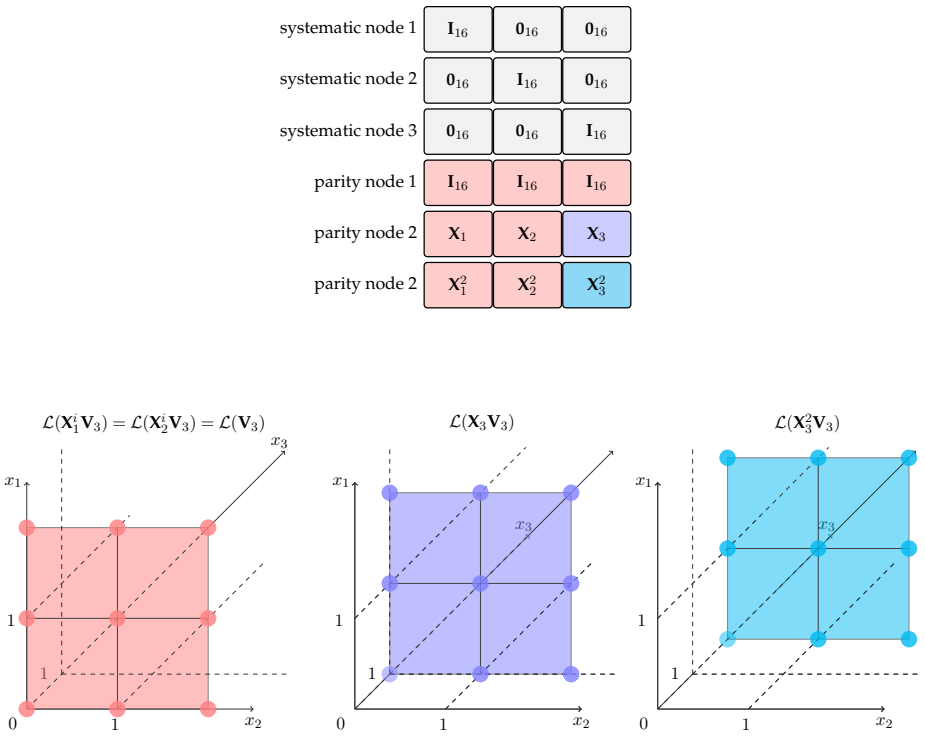


Figure 3.8: We illustrate some properties of the repair of node 3 in our $(6, 3)$ code. The red boxes in the coding matrix denote the matrices for which $\mathbf{V}_3$ is an invariant subspace. We also we depict the dots-on-a-lattice representation of the repair spaces. The dots of any of the products of $\mathbf{V}_3$ with $\mathbf{X}^{x_1}, \mathbf{X}_2^{x_2}$ have the same lattice representation, whereas $\mathbf{X}_3^{x_3}\mathbf{V}_3$, $x_3 \neq 0$, correspond to a disjoint sets of lattice points.

considered to belong into m^k groups of columns. Each column of a specific group has identical non-zero support with any other vector in that group. Then, any two columns within a block

$$\begin{bmatrix} \mathbf{I}_{m^k} \\ \lambda_{1,i}\mathbf{X}_i \\ \lambda_{2,i}\mathbf{X}_i^2 \\ \vdots \\ \lambda_{P-1,i}\mathbf{X}_i^{P-1} \end{bmatrix} \tag{3.100}$$

are orthogonal since their nonzero supports have zero overlap. Hence, a linear dependence will only exist among columns of a given non-zero support. We can

then rewrite the matrix determinant of (3.99) as

$$\det\left(\mathbf{P}\begin{bmatrix}\mathbf{B}_1 & \mathbf{0}_{m^k\times m^k} & \cdots & \mathbf{0}_{m^k\times m^k}\\ \mathbf{0}_{m^k\times m^k} & \mathbf{B}_2 & \cdots & \mathbf{0}_{m^k\times m^k}\\ \vdots & & & \\ \mathbf{0}_{m^k\times m^k} & \cdots & \mathbf{0}_{m^k\times m^k} & \mathbf{B}_P\end{bmatrix}\mathbf{P}\right) = |\mathbf{P}|^2\prod_{i=1}^{P}|\mathbf{B}_i|, \tag{3.101}$$

where $\mathbf{P}$ is the permutation matrix that groups the columns of the matrix according to their non-zero support, and $\mathbf{B}_i$ is a $P\times P$ full matrix of the form

$$\begin{bmatrix}\rho_{i_1,j_1}\lambda_{i_1,j_1} & \rho_{i_1,j_2}\lambda_{i_1,j_2} & \cdots & \rho_{i_1,j_P}\lambda_{i_1,j_P}\\ \rho_{i_2,j_1}\lambda_{i_2,j_1} & \rho_{i_2,j_2}\lambda_{i_2,j_2} & \cdots & \rho_{i_2,j_P}\lambda_{i_2,j_P}\\ \vdots & \vdots & \ddots & \vdots\\ \rho_{i_P,j_1}\lambda_{i_P,j_1} & \rho_{i_P,j_2}\lambda_{i_P,j_2} & \cdots & \rho_{i_P,j_P}\lambda_{i_P,j_P}\end{bmatrix}, \tag{3.102}$$

where ρ_{i_1,j_1} is some m^{th} root of unity, the indeces depend on i, and no $\lambda_{i,j}$ appears more than once within each matrix. We intend to show that the determinant of the above matrix is not the zero-polynomial. To do so, setting $\lambda_{i_l,j_l}=1, \lambda_{i_l,j_{l'}}=0$ for $l\neq l'$ will make the above matrix a diagonal matrix, whose determinant is $\prod_{l=1}^{P}\rho_{i_l,j_l}$ which is clearly not zero. Therefore, the polynomial formed by the above determinant cannot be the zero polynomial. Accordingly, we can compute the determinant of each DC in this way. In the same manner, each of them will be a nonzero polynomial in $\lambda_{i,j}$. The product of all these determinants will as well be a nonzero polynomial in $\lambda_{i,j}$ of some degree D. By the Schwartz-Zippel lemma, we know that when we draw $\lambda_{i,j}$ uniformly at random over a field of order q, this induced polynomial is zero with probability less than or equal to $\frac{D}{q}$. Hence, the MDS property is satisfied with probability arbitrarily close to 1, for sufficiently large finite fields, whose order is $q=m\cdot C+1$ and C is a free nonzero variable[8] that can scale to infinity.

3.9 Connection to Permutation-Matrix Based Codes

Here we investigate an interesting connection between our systematic-repair optimal codes of Section 3.8 and the permutation-matrix based codes presented in [99] and [17]. A similar connection was exploited under a subspace interference alignment framework in [17]. Under a similarity transformation, our codes

[8]Note that the existence of infinite primes q of the form $m\cdot C+1$ is guaranteed by Dirichlet's theorem on arithmetic progressions [36].

are equivalent to ones with coding matrices picked as specific permutation matrices. Multiplying the column space of an $\mathbf{X}_i$ matrix of our construction with the Hadamard matrix $\mathbf{H}_{m^k}$, yields a matrix that is a permutation of the columns of the Hadamard matrix

$$\mathbf{H}_{m^k}^{-1}\mathbf{X}_i\mathbf{H}_{m^k} = \mathbf{H}_{m^k}^{-1}\mathbf{H}_{m^k}\mathbf{P}_i = \mathbf{P}_i, \tag{3.103}$$

where $\mathbf{P}_i$ is some permutation matrix. This is due to the fact that the elements of $\mathcal{H}_{m^k}$ wrap around when multiplied with the $\mathbf{X}_i$ matrices. More precisely $\mathcal{L}(\mathbf{H}_{m^k}) = \mathcal{L}(\mathbf{X}_i\mathbf{H}_{m^k})$ for any i. Hence, we rewrite the $\mathbf{A}^{(m,k)}$ matrix of (3.91) as

$$\begin{aligned}
&\left(\mathbf{I}_m \otimes \mathbf{H}_{m^k}^{-1}\right)\mathbf{A}^{(k,m)}\left(\mathbf{I}_k \otimes \mathbf{H}_{m^k}\right)\\
&= \left(\mathbf{I}_m \otimes \mathbf{H}_{m^k}^{-1}\right)\begin{bmatrix} \mathbf{I}_{m^k}\mathbf{H}_{m^k} & \mathbf{I}_{m^k}\mathbf{H}_{m^k} & \dots & \mathbf{I}_{m^k}\mathbf{H}_{m^k} \\ \lambda_{1,1}\mathbf{X}_1\mathbf{H}_{m^k} & \lambda_{1,2}\mathbf{X}_2\mathbf{H}_{m^k} & \dots & \lambda_{1,k}\mathbf{X}_k\mathbf{H}_{m^k} \\ \lambda_{2,1}\mathbf{X}_1^2\mathbf{H}_{m^k} & \lambda_{2,2}\mathbf{X}_2^2\mathbf{H}_{m^k} & \dots & \lambda_{2,k}\mathbf{X}_k^2\mathbf{H}_{m^k} \\ \vdots & & & \\ \lambda_{m-1,k}\mathbf{X}_1^{m-1}\mathbf{H}_{m^k} & \lambda_{m-1,2}\mathbf{X}_2^{m-1}\mathbf{H}_{m^k} & \dots & \lambda_{m-1,k}\mathbf{X}_k^{m-1}\mathbf{H}_{m^k} \end{bmatrix}\\
&= \left(\mathbf{I}_m \otimes \mathbf{H}_{m^k}^{-1}\right)\begin{bmatrix} \mathbf{H}_{m^k} & \mathbf{H}_{m^k} & \dots & \mathbf{H}_{m^k} \\ \lambda_{1,1}\mathbf{H}_{m^k}\mathbf{P}_{1,1} & \lambda_{1,2}\mathbf{H}_{m^k}\mathbf{P}_{1,2} & \dots & \lambda_{1,k}\mathbf{H}_{m^k}\mathbf{P}_{1,k} \\ \lambda_{2,1}\mathbf{H}_{m^k}\mathbf{P}_{2,1} & \lambda_{2,2}\mathbf{H}_{m^k}\mathbf{P}_{2,2} & \dots & \lambda_{2,k}\mathbf{H}_{m^k}\mathbf{P}_{2,k} \\ \vdots & & & \\ \lambda_{m-1,k}\mathbf{H}_{m^k}\mathbf{P}_{m-1,1} & \lambda_{m-1,2}\mathbf{H}_{m^k}\mathbf{P}_{m-1,2} & \dots & \lambda_{m-1,k}\mathbf{H}_{m^k}\mathbf{P}_{m-1,k} \end{bmatrix}\\
&= \begin{bmatrix} \mathbf{I}_{m^k} & \mathbf{I}_{m^k} & \dots & \mathbf{I}_{m^k} \\ \lambda_{1,1}\mathbf{P}_{1,1} & \lambda_{1,2}\mathbf{P}_{1,2} & \dots & \lambda_{1,k}\mathbf{P}_{1,k} \\ \lambda_{2,1}\mathbf{P}_{2,1} & \lambda_{2,2}\mathbf{P}_{2,2} & \dots & \lambda_{2,k}\mathbf{P}_{2,k} \\ \vdots & & & \\ \lambda_{m-1,k}\mathbf{P}_{m-1,1} & \lambda_{m-1,2}\mathbf{P}_{m-1,2} & \dots & \lambda_{m-1,k}\mathbf{P}_{m-1,k} \end{bmatrix},
\end{aligned} \tag{3.104}$$

where $\mathbf{P}_{i,j}$ is a permutation matrix. The systematic nodes of this equivalent $(k+m,k)$-MDS code can be optimally repaired using the repair matrices $\mathbf{V}_i\mathbf{H}_i^{-1}$, where $\mathbf{V}_i$ has the columns of the set $\mathcal{V}_i = \left\{\prod_{s=1,s\neq i}^{k}\mathbf{X}_s^{x_s}\mathbf{w} : x_s \in \{0,1,\dots,m-1\}\right\}$. This is true since the rank properties of the corresponding useful and interference spaces are invarianteto full-rank column transformations. Interestingly, this connection is two-way.

We find the connection manifested by the above equivalence examples to be of specific interest. Further investigation on it may lead to more understanding of the repair optimal high-rate designs. For the $\frac{k}{n} \leq \frac{1}{2}$ rate regime, there are several results and explicit codes for all parameters (n,k,d) where interference alignment has been fundamental in these solutions [81, 89, 95]. Formulation of an interference alignment framework for optimally repairing all nodes of a high-rate MDS code remains an interesting open problem.

3.10 Conclusions

We presented the first explicit, high-rate, $(k+2,k)$ erasure MDS storage code that achieves optimal repair bandwidth for any single node failure, including the parities. Our construction is based on perfect interference alignment properties offered by Hadamard designs. Moreover, we generalize our constructions to erasure codes with m-parities that achieve optimally repair of the systematic parts. Interestingly, the size of the code for the high-rate regime is exponential in the parameter k. Fundamental limits on the size of the code for a given repair bandwidth is an interesting and ongoing area of future work [19].

3.11 Proof of Lemma 1

It is easy to show that the matrices are orthogonal [98]. Observe that $\mathbf{H}_N = \mathbf{H}_N^T$ and

$$\begin{aligned}\mathbf{H}_N\mathbf{H}_N^T = \mathbf{H}_N\mathbf{H}_N = \begin{bmatrix} 2\mathbf{H}_{\frac{N}{2}}\mathbf{H}_{\frac{N}{2}} & \mathbf{0}_{\frac{N}{2}\times\frac{N}{2}} \\ \mathbf{0}_{\frac{N}{2}\times\frac{N}{2}} & 2\mathbf{H}_{\frac{N}{2}}\mathbf{H}_{\frac{N}{2}} \end{bmatrix} &= 2\left(\mathbf{I}_2 \otimes \mathbf{H}_{\frac{N}{2}}\mathbf{H}_{\frac{N}{2}}\right) \\ &= 2\left(\mathbf{I}_2 \otimes 2\left(\mathbf{I}_2 \otimes \mathbf{H}_{\frac{N}{4}}\mathbf{H}_{\frac{N}{4}}\right)\right) \\ &= 4\left(\mathbf{I}_4 \otimes \mathbf{H}_{\frac{N}{4}}\mathbf{H}_{\frac{N}{4}}\right) \\ &\vdots \\ &= N\cdot(\mathbf{I}_N \otimes \mathbf{H}_1\mathbf{H}_1) = N\cdot\mathbf{I}_N. \quad (3.105)\end{aligned}$$

We also have that $N \neq 0 \pmod q$, for $q > 2$, thus the rank of $\mathbf{H}_N$ is N and its columns are mutually orthogonal. Then, let an $N \times N$ diagonal matrix

$$\mathbf{X}_i = \mathbf{I}_{2^{i-1}} \otimes \text{blkdiag}\left(\mathbf{I}_{\frac{N}{2^i}}, -\mathbf{I}_{\frac{N}{2^i}}\right) \quad (3.106)$$

defined for $i = [\log_2(N)]$. $\mathbf{X}_i$ is a diagonal matrix, whose elements is a series of alternating 1s and -1s, starting with $\frac{N}{2^i}$ 1s that flip to -1s and back every $\frac{N}{2^i}$ positions. We can now expand $\mathbf{H}_N$ in the following way

$$\mathbf{H}_N = \begin{bmatrix} \mathbf{H}_{\frac{N}{2}} & \mathbf{H}_{\frac{N}{2}} \\ \mathbf{H}_{\frac{N}{2}} & -\mathbf{H}_{\frac{N}{2}} \end{bmatrix} = \begin{bmatrix} \underbrace{\mathbf{1}_{2\times1} \otimes \mathbf{H}_{\frac{N}{2}}}_{\mathbf{F}_1} & \mathbf{X}_1\left(\mathbf{1}_{2\times1} \otimes \mathbf{H}_{\frac{N}{2}}\right) \end{bmatrix}. \quad (3.107)$$

We proceed in the same manner by expanding all "smaller" $\mathbf{H}_{\frac{N}{2^i}}$s

$$\begin{aligned}
\mathbf{F}_1 =& \mathbf{1}_{2\times 1} \otimes \left[\mathbf{1}_{2\times 1} \otimes \mathbf{H}_{\frac{N}{2^2}} \quad \mathbf{X}_2 \left(\mathbf{1}_{2\times 1} \otimes \mathbf{H}_{\frac{N}{2^2}}\right)\right] = \left[\underbrace{\mathbf{1}_{2^2\times 1} \otimes \mathbf{H}_{\frac{N}{2^2}}}_{\mathbf{F}_2} \quad \mathbf{X}_3 \left(\mathbf{1}_{2^2\times 1} \otimes \mathbf{H}_{\frac{N}{2^2}}\right)\right] \\
\mathbf{F}_2 =& \left[\underbrace{\mathbf{1}_{2^3\times 1} \otimes \mathbf{H}_{\frac{N}{2^3}}}_{\mathbf{F}_3} \quad \mathbf{X}_3 \left(\mathbf{1}_{2^3\times 1} \otimes \mathbf{H}_{\frac{N}{2^3}}\right)\right] \\
&\vdots \\
\mathbf{F}_{\log_2(N)-1} =& \left[\mathbf{1}_{N\times 1} \quad \mathbf{X}_{\log_2(N)}\mathbf{1}_{N\times 1}\right], \qquad (3.108)
\end{aligned}$$

where $\mathbf{F}_i$ is an $N \times \frac{N}{2^i}$ matrix and $\mathbf{F}_{\log_2(N)} = \mathbf{w}$ is the all ones vector of length N. Thus,

$$\begin{aligned}
\text{span}\left(\mathbf{H}_N\right) &= \text{span}\left(\left[\mathbf{F}_1 \quad \mathbf{X}_1\mathbf{F}_1\right]\right) = \text{span}\left(\left[\mathbf{F}_2 \quad \mathbf{X}_2\mathbf{F}_2 \quad \mathbf{X}_1\mathbf{F}_2 \quad \mathbf{X}_1\mathbf{X}_2\mathbf{F}_2\right]\right) \\
&\vdots \\
&= \text{span}\left(\left\{\left(\prod_{i=1}^{\log_2(N)} \mathbf{X}_i^{x_i}\right)\mathbf{w} : x_i \in \{0,1\}\right\}\right), \qquad (3.109)
\end{aligned}$$

which proves the final part of Lemma 1. □

Chapter 4

Locally Repairable Codes

In this chapter, we explore the repair metric of *locality*, which corresponds to the number of disk accesses required during a single node repair. Under this metric we characterize an information theoretic trade-off that binds together locality, code distance, and the storage capacity of each node. We show the existence of optimal *locally repairable codes* (LRCs) that achieve this trade-off.

The achievability proof uses a locality aware flow-graph gadget which leads to a randomized code construction. Finally, we present an optimal and explicit LRC that achieves arbitrarily high data-rates. Our locality optimal construction is based on simple combinations of Reed-Solomon blocks.[1]

4.1 Introduction

Consider a code of length n, with k information symbols. A symbol i has locality r_i, if it can be reconstructed by accessing r_i other symbols in the code. For example, in an (n, k) maximum-distance separable (MDS) code, every symbol has trivial locality k. We will say that a systematic code has *information-symbol locality* r, if all the k information symbols have locality r. Similarly, a code has *all-symbol locality* r, if all n symbols have locality r.

Different repair metrics optimize alternative objectives which may be useful in various storage systems depending on the specific architectures and workloads. Locality allows repairs by communicating with a very small subset of nodes. However, codes with small locality are suboptimal in terms of the repair bandwidth and disk-I/O metrics. Further, as we show in this chapter, LRCs must either sacrifice some code distance, or use more storage compared to MDS codes to achieve low

[1]Contributions Statement: Most parts of this chapter appear in [76]. Prof. Alex Dimakis supervised the project and all coauthors had equal contribution to this work.

locality. A recent alternative family of storage codes that seems to be practically applicable and offers higher storage efficiency and small repair bandwidth was proposed in [83]. One important benefit of codes with small locality is that their simple designs are easily implementable in distributed file systems like Hadoop [88] and Windows Azure Storage [51]. Further, codes with low locality were recently deployed in production clusters [51] and operating systems like Windows Server and Windows 8.1 [52].

Codes with small locality were initially introduced in [46, 50]. Gopalan *et al.* [44] pioneered the theoretical study of locality by discovering a trade-off between code distance and information-symbol locality. In [44] the trade-off was obtained for scalar linear codes, *i.e.*, codes where each source and coded symbol is represented by a scalar over some finite field, and the each coded symbol is a linear function of the source symbols. Bounds on the code-distance for a given locality as well as code constructions were presented in parallel and subsequent works [59, 80, 85, 100]. Some works extend the designs and distance bounds to the case where repair bandwidth and locality are jointly optimized, under multiple local failures [59, 85], and under security constraints [85].

Our Contributions: We generalize the prior work of [44] and provide a distance bound that is universal: it holds for both linear and nonlinear codes, while it allows both scalar and vector code designs, where input and output symbols can have arbitrary sizes. We proceed to show that this information theoretic trade-off is achievable, when $r+1$ divides the length of the code n. We conclude with presenting explicit constructions for codes with all-symbol locality. We provide a formal definition of an LRC and then proceed with stating our three contributions in more detail.

Definition 1. *An (n, r, d, M, α)-LRC is a code that takes a file of size M bits, encodes it in n coded symbols of size α bits, and any of these n coded symbols can be reconstructed by accessing and processing at most r other symbols. Moreover, the minimum-distance of the code is d, i.e., the file of size M can be reconstructed by accessing any $n-d+1$ of the n coded symbols.*[2]

[2]In comparison, the definition of an information-symbol (or all-symbol) locality code in [44] assumes that the encoding is a linear mapping from k input to n output symbols. Moreover, the input and output symbols are assumed to be of the same size, i.e., of the same number of bits. Our defi-

Our three contributions follow:
1) An information theoretic bound on code distance d: We present a bound that binds together the code distance d, the locality r, and the size of each coded symbol α (*i.e.*, the storage capacity of each node). The bound is information theoretic and covers all codes, linear or nonlinear, and reads as follows:

Theorem 1. *An (n, r, d, M, α)-LRC, as defined above, has distance d that is bounded as*

$$d \leq n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{r\alpha} \right\rceil + 2.$$

We establish our bound using an impossibility result for values of distance d larger than the above. The impossibility result uses an algorithmic proof similar to [44] and counting arguments on the entropy of subsets of coded symbols. We would like to note that when we set $M = k$ and $\alpha = 1$, which corresponds to the scalar-code regime, we obtain the same bound as [44], that is

$$d \leq n - k - \left\lceil \frac{k}{r} \right\rceil + 2.$$

2) Achievability of the distance bound when $(r + 1)$ divides n:

Theorem 2. *Let $(r + 1)|n$ and $r \leq n - d$. Then, there exist (n, r, d, M, α)-LRCs with minimum code distance*

$$d = n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{ra} \right\rceil + 2,$$

over a sufficiently large finite field.

We prove the achievability using a novel information flow-graph gadget, in a similar manner to [34]. In contrast to [34], the flow-graph that we construct is finite, locality aware, and simpler to analyze. The existence of (n, r, d, M, α)-LRCs is established through a capacity achieving scheme on a multicast network [47], specified by the aforementioned flow-graph. The obtained LRCs are vector codes: codes where each source and coded symbol is represented as a vector (not necessarily

nition is more general: both linear and non-linear codes are allowed, and the size of the input and output symbols can be different.

of the same length). This is yet another case where vector codes are employed to achieve an optimal trade-off. In [34], the codes achieving the optimal repair bandwidth-storage trade-off are also vector linear.

3) Explicit code constructions of optimal LRCs: We construct explicit LRCs for the following set of coding parameters:

$$\left(n, r, d = n - k + 1, M, \alpha = \frac{r+1}{r} \cdot \frac{M}{k}\right), \text{ such that } (r+1)|n.$$

Our codes are optimal when $(r+1) \nmid k$. The above parameters correspond to codes with all-symbol locality r and rate $\left(1 + \frac{1}{r}\right) \cdot \frac{k}{n}$, where any k coded symbols suffice to recover the file. Our designs are vector-linear and each symbol stored requires only $r \cdot O(\log(n))$ bits in its representation. We show that these codes not only have optimal locality, but also admit simple node repairs based on XORs.

The remainder of this chapter is organized as follows. In Section 4.2, we provide the coding theoretic definitions used in the subsequent sections. In Section 4.3, we provide a distance bound for codes with all-symbol locality. In Section 4.4, we prove that this bound is achievable using random vector codes. In Section 4.5, we provide an explicit LRC construction, and discuss its properties.

4.2 Preliminaries

A way to calculate the code distance of a linear code is through its generator matrix: calculating the minimum distance is equivalent to finding the largest set of columns of the generator matrix that are not full-rank [44, 59]. In the following, we use entropy to characterize the distance of a code. This is the key difference to the related works in [44], [59], which provide results only for linear codes. The use of the entropy of coded symbols ensures that our bounds are universal: they hold for linear and nonlinear codes, for any file and coded symbol size, irrespective of a vector or scalar representation. The main properties that we exploit here are the following: entropy is oblivious to the encoding process (linear or nonlinear), it can accommodate different input or output symbol sizes, and different symbol representations (scalar or vector). We will now proceed with our technical discussion.

Let a file of size M bits[3] be represented as an M-dimensional vector $\mathbf{x}$,

[3]The M file elements can also be elements of an appropriate q-ary alphabet, for any $q \geq 2$. We

whose elements can be considered as independent and identically distributed (i.i.d.) uniform random variables, each drawn from a Galois Field $\mathbb{GF}(2)$, referred to as $\mathbb{F}_2$ for convenience.[4] The (binary) entropy of $\mathbf{x}$ will then be[5]

$$H(\mathbf{x}) = M. \tag{4.1}$$

Moreover, let $G : \mathbb{F}_2^M \mapsto \mathbb{F}_2^{n \cdot \alpha}$ be an encoding (generator) function, that takes as input the file $\mathbf{x}$ and maps it to n coded symbols, each of size α:

$$G(\mathbf{x}) = \mathbf{y} = [Y_1 \dots Y_n]$$

where each encoded symbol has entropy

$$H(Y_i) \leq \alpha,$$

for all $i \in [n]$, where $[n] = \{1, \dots, n\}$. In the following, we frequently refer to α as the storage cost per coded symbol.

The generator function G defines an n-length code $\mathcal{C}$. The effective data rate of the code is the ratio of the total source entropy to the aggregate entropy of the stored encoded information

$$R = \frac{H(\mathbf{x})}{\sum_{i=1}^{n} H(Y_i)}.$$

We continue with a definition for the minimum code distance.

Definition 2 (Minimum code distance)**.** *The minimum distance d of the code $\mathcal{C}$ is equal to the minimum number of erasures of coded symbols in $\mathbf{y}$ after which the entropy of the non-erased symbols is strictly less than M, that is,*

$$d = \min_{H(\{Y_1,\dots,Y_n\} \setminus \mathcal{E}) < M} |\mathcal{E}|$$

where $\mathcal{E} \in 2^{\{Y_1,\dots,Y_n\}}$ and $2^{\{Y_1,\dots,Y_n\}}$ is the power set of the symbols in $\{Y_1, \dots, Y_n\}$.

keep the discussion in bits for simplicity.

[4] We assume that $\mathbf{x}$ consists of i.i.d. uniform random variables, since all M bits are assumed to hold the same amount of useful information (are of equal entropy).

[5] If the base alphabet was q-ary instead of binary, then we would need to use q-ary entropies.

In other words, when a code has minimum distance d, this means that there is sufficient entropy after any $d-1$ erasures of coded symbols to reconstruct the file. The above definition can be restated in its dual form: the minimum distance d of the code $\mathcal{C}$ is equal to the length of the code n, minus the maximum number of coded symbols in $\mathbf{y}$ that cannot reconstruct the file, that is,

$$d = n - \max_{H(\mathcal{S})<M} |\mathcal{S}|$$

where $\mathcal{S} \in 2^{\{Y_1,\ldots,Y_n\}}$.

Remark 5. *Observe that the above distance definition applies to linear, or nonlinear codes, and to any length of input and output symbols.*

We continue with the definition of repair locality.

Definition 3 (Repair Locality)**.** *A coded symbol Y_i, $i \in [n]$, is said to have repair locality r, if there exists at least one set of coded symbols with indices in $\mathcal{R}(i) \subseteq [n]\backslash\{i\}$, call it $Y_{\mathcal{R}(i)}$, of cardinality $|\mathcal{R}(i)| = r$, and a function $g_i : \mathbb{F}_2^{r\cdot\alpha} \to \mathbb{F}_2^{\alpha}$, such that Y_i can be expressed as a function of these r coded symbols, i.e., $Y_i = g_i(Y_{\mathcal{R}(i)})$.*

4.3 A Universal bound between code distance, locality, and storage cost

In this section, we provide an information theoretic bound for locally repairable codes. Specifically, we answer the question: what is the maximum possible distance d of a code that has locality r? We provide a universal upper bound on the minimum distance of a code of length n, with all-symbol locality r, where each coded symbol has size α. We do so by an algorithmic proof, in a similar manner to [44]. Deriving such a distance bound reduces to lower bounding the cardinality of the largest set $\mathcal{S}$ of coded symbols whose entropy is less than M.

In our proof, the only structural property that we use, is the fact that every coded symbol has locality r. Specifically, if a code $\mathcal{C}$ has locality r, then for each of its coded symbols, say Y_i, there exist at least one group of at most r other coded symbols $Y_{\mathcal{R}(i)}$ that can reconstruct Y_i, for $i \in [n]$. We define as

$$\Gamma(i) = \{i, \mathcal{R}(i)\}$$

a set of $r+1$ coded symbols that has the property

$$H(Y_{\Gamma(i)}) = H(Y_i, Y_{\mathcal{R}(i)}) = H(Y_{\mathcal{R}(i)}) \leq r\alpha,$$

for all $i \in [n]$; the above comes due to the functional dependencies induced by locality. We refer to such a set of coded symbols as an $(r+1)$-*group*. The theorem and its proof follow.

Theorem 2. *An (n, r, d, M, α)-LRC has minimum distance d that is bounded as*

$$d \leq n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{r\alpha} \right\rceil + 2.$$

Proof. In this proof we use some of the algorithmic techniques that were introduced in [44]. Our aim is to lower bound the cardinality of a set $\mathcal{S}$, consisting of the *maximum* number of coded symbols with entropy $H(\mathcal{S})$ *strictly less* than the filesize M. This bound will be equivalent to an upper bound on the minimum code distance d, since

$$d = n - \max_{\substack{\mathcal{S} \subset \{Y_1, \ldots, Y_n\} \\ H(\mathcal{S}) < M}} |\mathcal{S}|.$$

To build such a maximally sized set described above, we need to collect as many symbols as possible that have as small joint entropy as possible. Subsets of coded symbols that have many dependencies (small joint entropy) are preferred to subsets of the the same cardinality, but of larger joint entropy. The only structural information about the code that we can exploit to introduce dependencies is that of repair locality: every repair group $Y_{\Gamma(i)}$ has joint entropy at most $r \cdot \alpha$, while an arbitrary set of $r+1$ symbols can have joint entropy up to $(r+1) \cdot \alpha$.

We build the set $\mathcal{S}$ in an algorithmic way through iterative steps. The algorithm picks as many $(r+1)$-groups as possible, until it exits. The algorithm that builds the set follows in Fig. 4.1. We proceed with analyzing the size and entropy of the sets that it can possibly construct. The goal of our analysis is to lower bound the size of the set $\mathcal{S}_l$ that the algorithm can possibly produce. This will tell us that no matter how the code is constructed, its minimum distance cannot be more than $n - |\mathcal{S}_l|$.

We denote the collection of coded symbols at each step of the iteration as $\mathcal{S}_i$. At each step i, the difference in cardinality between $\mathcal{S}_i$ and $\mathcal{S}_{i-1}$ is denoted as

$$s_i = |\mathcal{S}_i| - |\mathcal{S}_{i-1}| \tag{4.2}$$

step	
1	Set $\mathcal{S}_0 = \emptyset$ and $i = 1$
2	`WHILE` $H(\mathcal{S}_{i-1}) < M$
3	Pick a coded symbol $Y_j \notin \mathcal{S}_{i-1}$
4	`IF` $H(\mathcal{S}_{i-1} \cup \{Y_{\Gamma(j)}\}) < M$
5	`set` $\mathcal{S}_i = \mathcal{S}_{i-1} \cup Y_{\Gamma(j)}$
6	`ELSE IF` $H(\mathcal{S}_{i-1} \cup \{Y_{\Gamma(j)}\}) \geq M$
7	$\mathcal{T} = \underset{\mathcal{T}' \subset \Gamma(j); H(Y_{\mathcal{T}'} \cup \mathcal{S}_{i-1}) < M}{\arg\max} \lvert\mathcal{T}'\rvert$
8	`IF` $\mathcal{T} = \emptyset$
9	`EXIT`
10	`ELSE`
11	`set` $S_i = S_{i-1} \cup Y_{\mathcal{T}}$
12	`EXIT`
13	$i = i + 1$

Figure 4.1: The algorithm that builds set $\mathcal{S}$.

and the difference between the entropy of the two sets as

$$h_i = H(\mathcal{S}_i) - H(\mathcal{S}_{i-1}). \tag{4.3}$$

The algorithm exits before reaching $H(\mathcal{S}_i) \geq M$. There are two ways that the algorithm terminates:
i) it either collects $(r+1)$-groups until it exits at line 9, or
ii) the last subset of coded symbols that is added to $\mathcal{S}_{i-1}$ is smaller than $r+1$ and the algorithm exits at line 12, after collecting some subset of an $(r+1)$-group, such that $H(\mathcal{S}_i) < M$ is not violated.
Let us denote by l the *last* iteration of the algorithm during which a new *non-empty* set of coded symbols is added to the current set of coded symbols. We shall now proceed with lower bounding $|\mathcal{S}_l|$.

Case *i) The algorithm exits at line* 9*:*
Since the algorithm exits at 9, this means that its last iteration is the $(l+1)$-st, where no more symbols are added. Again, we denote by l the *last* iteration during which our set of coded symbols is expanded by a non-empty set. First observe that, for any $1 \leq i \leq l$, we have

$$1 \leq s_i \leq r+1 \tag{4.4}$$

since at each iteration the algorithm augments the set $\mathcal{S}_{i-1}$ by at least one new symbol, *i.e.*, Y_j, which is always possible since $H(\mathcal{S}_{i-1}) < M$, for all $i \leq l$ and $H(Y_1, \ldots, Y_n) = M$. Then, $s_i \leq r + 1$ is a consequence of the fact that

$$|\mathcal{S}_i| = |\mathcal{S}_{i-1} \cup Y_{\Gamma(i)}| \leq |\mathcal{S}_{i-1}| + |Y_{\Gamma(i)}| \leq |\mathcal{S}_{i-1}| + r + 1.$$

We also have that

$$h_i \leq (s_i - 1)\alpha. \tag{4.5}$$

To see why the above is true, let $\mathcal{S}_{i-1} = \mathcal{A} \cup \mathcal{B}$, where $\mathcal{B} = \mathcal{S}_{i-1} \cap Y_{\mathcal{R}(j)}$ is the subset of symbols from $Y_{\mathcal{R}(j)}$ that are already in $\mathcal{S}_{i-1}$ ($\mathcal{B}$ can be empty if no symbols from $\mathcal{R}(j)$ are in $\mathcal{S}_{i-1}$). Then,

$$\begin{aligned} H(\mathcal{S}_i) = H(\mathcal{S}_{i-1} \cup Y_{\Gamma(j)}) &= H(\mathcal{S}_{i-1} \cup \{Y_{\mathcal{R}(j)} \backslash \mathcal{B}\}) \leq H(\mathcal{S}_{i-1}) + H(Y_{\mathcal{R}(j)} \backslash \mathcal{B}) \\ &\leq H(\mathcal{S}_{i-1}) + |Y_{\mathcal{R}(j)} \backslash \mathcal{B}|\alpha \\ &= H(\mathcal{S}_{i-1}) + (s_i - 1)\alpha, \end{aligned} \tag{4.6}$$

where the second equality comes from the fact that Y_j is a function of some symbols in $\mathcal{S}_{i-1} \cup \{Y_{\mathcal{R}(j)} \backslash \mathcal{B}\}$, due to locality, and the last equality is due to

$$s_i = |\mathcal{S}_i| - |\mathcal{S}_{i-1}| = |Y_{\Gamma(j)} \backslash \mathcal{B}| = |Y_{\mathcal{R}(j)} \backslash \mathcal{B}| + 1.$$

From (4.5), we also obtain

$$\alpha \cdot s_i \geq h_i + \alpha. \tag{4.7}$$

Now, we can start bounding the size of $\mathcal{S}_l$ as follows

$$\alpha|\mathcal{S}_l| = \alpha \sum_{i=1}^{l} s_i \overset{(4.7)}{\geq} \sum_{i=1}^{l} (h_i + \alpha) = \left(\sum_{i=1}^{l} h_i \right) + l \cdot \alpha = H(\mathcal{S}_l) + l \cdot \alpha. \tag{4.8}$$

We continue with lower bounding the two quantities in (4.8): $H(\mathcal{S}_l)$ and $l \cdot \alpha$. First observe that since the algorithm is exiting, it means that the aggregate entropy $H(\mathcal{S}_l) = \sum_{i=1}^{l} h_i$ is so large that no other symbol can be added to our current set $\mathcal{S}_l$, without violating the entropy condition. Hence,

$$H(\mathcal{S}_l) \geq M - \alpha. \tag{4.9}$$

Assume otherwise, *i.e.*, for example $H(\mathcal{S}_l) \leq M - \alpha - \epsilon$, for any $\epsilon > 0$. Then, any coded symbol not in $\mathcal{S}_l$ can be added in $\mathcal{S}_l$ so that the aggregate entropy is at most

$M - \epsilon$: the new symbol can only increase the joint entropy by at most α. Hence, $H(\mathcal{S}_l)$ has to be at least $M - \alpha$.

Now we will lower bound l, the number of iterations to reach an entropy of at least $M - \alpha$. Since the algorithm is assumed to exit at line 9, as mentioned before, at every iteration i we have $s_i \leq r + 1$ and $h_i \leq (s_i - 1)\alpha$, for all $1 \leq i \leq l$. The minimum number of iterations occurs, when at each iteration the algorithm picks sets such that the entropy increase h_i is equal to its upper bound $r \cdot \alpha$. Therefore,

$$l \geq \left\lceil \frac{H(S_l)}{r \cdot \alpha} \right\rceil \geq \left\lceil \frac{M - \alpha}{r \cdot \alpha} \right\rceil. \tag{4.10}$$

Using (4.9) and (4.10), we can rewrite (4.8) as

$$\begin{aligned}
&\alpha|\mathcal{S}_l| \geq H(\mathcal{S}_l) + l \cdot \alpha \geq M - \alpha + \alpha \cdot \left\lceil \frac{M - \alpha}{r \cdot \alpha} \right\rceil \\
\Rightarrow &|\mathcal{S}_l| \geq \left\lceil \frac{M - \alpha + \alpha \cdot \left\lceil \frac{M-\alpha}{r\cdot\alpha} \right\rceil}{\alpha} \right\rceil = \left\lceil \frac{M}{\alpha} - 1 + \left\lceil \frac{M - \alpha}{r \cdot \alpha} \right\rceil \right\rceil \\
&\overset{(i)}{=} \left\lceil \frac{M}{\alpha} \right\rceil - 1 + \left\lceil \frac{M - \alpha}{r \cdot \alpha} \right\rceil = \left\lceil \frac{M}{\alpha} \right\rceil - 1 + \left\lceil \frac{M}{r \cdot \alpha} - \frac{1}{r} \right\rceil \\
&\geq \left\lceil \frac{M}{\alpha} \right\rceil - 1 + \left\lceil \frac{M}{r \cdot \alpha} \right\rceil - 1 \\
\Rightarrow &d \leq n - |S_l| \leq n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{r \cdot \alpha} \right\rceil + 2,
\end{aligned} \tag{4.11}$$

where the equality in (i) comes from the fact that $\lceil x + n \rceil = \lceil x \rceil + n$, for any real number x and any integer n [63]; in our case $x = \frac{M}{\alpha}$ and $n = -1 + \left\lceil \frac{M-\alpha}{r\cdot\alpha} \right\rceil$.

Case *ii) The algorithm exits at line* 12*:*

In this case, the algorithm runs for l iterations; during the $l - 1$ first iterations, the algorithm augments $\mathcal{S}_{i-1}$ at step $1 \leq i \leq l - 1$, by entire $(r + 1)$-groups. During the last step $i = l$, the algorithm augments $\mathcal{S}_{l-1}$, by a *partial* subset of $Y_{\Gamma(j)}$, for some coded symbol Y_j not in $\mathcal{S}_{l-1}$. From the above, we get the following bounds

$$s_i \leq r + 1, \text{ for all } 1 \leq i \leq l - 1, \text{ and } s_l \leq r, \tag{4.12}$$

and

$$h_i \leq (s_i - 1)\alpha, \text{ for all } 1 \leq i \leq l - 1, \text{ and } h_l \leq s_l\alpha. \tag{4.13}$$

The right most part of the above bounds comes from the fact that, during the last iteration, at most r coded symbols can be added to the set $\mathcal{S}_{l-1}$. Moreover, in contrast to all other iterations, during the last iteration it is possible to augment $\mathcal{S}_{l-1}$ by s_l new coded symbols, all being independent to each other and any other symbols in $\mathcal{S}_{l-1}$; that is h_l can be as large as $s_l\alpha$.

We will again bound the size of $\mathcal{S}_l$, the maximal set of coded symbols that has entropy less than M. We use (4.13) and sum over all s_i's to obtain our bound on the size of $\mathcal{S}_l$:

$$\begin{aligned}
&\sum_{i=1}^{l} h_i \overset{(4.13)}{\leq} \left(\sum_{i=1}^{l-1} s_i\alpha\right) - (l-1)\cdot\alpha + s_l\cdot\alpha \\
\Rightarrow &\alpha\cdot\sum_{i=1}^{l} s_i \geq \sum_{i=1}^{l} h_i + (l-1)\cdot\alpha \Rightarrow \alpha\cdot|\mathcal{S}_l| \geq \sum_{i=1}^{l} h_i + (l-1)\cdot\alpha \\
\Rightarrow &\alpha\cdot|\mathcal{S}_l| \geq H(\mathcal{S}_l) + (l-1)\cdot\alpha. \qquad (4.14)
\end{aligned}$$

We now need to bound again the two quantities that control the bound in (4.14): $H(\mathcal{S}_l)$ and l. We can use the same bound as used in Case *i)* for $H(\mathcal{S}_l) = \sum_{i=1}^{l} h_i$, *i.e.*, the entropy of the constructed set $H(\mathcal{S}_l)$ has to be large enough, so that another iteration cannot be carried on:

$$H(\mathcal{S}_l) \geq M - \alpha. \qquad (4.15)$$

Again, let us assume otherwise: $H(\mathcal{S}_l) \leq M - \alpha - \epsilon$, for any $\epsilon > 0$. Then, any coded symbol not in $\mathcal{S}_l$ can be added in $\mathcal{S}_l$ so that the aggregate entropy is at most $M - \epsilon$. Hence, $H(\mathcal{S}_l) \geq M - \alpha$.

Now we will bound the number of iterations l. The last added subset of symbols $\mathcal{T}$ that augments $\mathcal{S}_{l-1}$ has cardinality less than, or equal to r. Otherwise, if $\mathcal{T}$ was an entire $(r+1)$-group, then the statement in line 6 of the algorithm would have been FALSE. This means that adding $(r+1)$-groups for all iterations, including the l-th one, can have as much entropy as $r\cdot l\cdot\alpha$, which has to be *at least* as much as M, or else we would not have been under *Case ii)* of the algorithm. Hence,

$$r\cdot l\cdot\alpha \geq M \Rightarrow l \geq \left\lceil \frac{M}{r\cdot\alpha} \right\rceil. \qquad (4.16)$$

We can now use the bounds in (4.15) and (4.16) to rewrite (4.14) as

$$\begin{aligned}
&\alpha|\mathcal{S}_l| \geq H(\mathcal{S}_l) + (l-1)\cdot\alpha \geq M - \alpha + \alpha\cdot\left(\left\lceil\frac{M}{r\cdot\alpha}\right\rceil - 1\right)\\
\Rightarrow &|\mathcal{S}_l| \geq \left\lceil\frac{M - 2\alpha + \alpha\cdot\left\lceil\frac{M}{r\cdot\alpha}\right\rceil}{\alpha}\right\rceil = \left\lceil\frac{M}{\alpha} - 2 + \left\lceil\frac{M}{r\cdot\alpha}\right\rceil\right\rceil \overset{(i)}{=} \left\lceil\frac{M}{\alpha}\right\rceil - 2 + \left\lceil\frac{M}{r\cdot\alpha}\right\rceil\\
\Rightarrow &d \leq n - |S_l| \leq n - \left\lceil\frac{M}{\alpha}\right\rceil - \left\lceil\frac{M}{r\cdot\alpha}\right\rceil + 2, \qquad (4.17)
\end{aligned}$$

where (i) comes from the fact that $\lceil x+n\rceil = \lceil x\rceil + n$, for any real number x and any integer n [63]. The bounds of (4.11) and (4.17) establish our theorem. □

Remark 6. *Observe that when $(r+1)|n$, we can partition the set of n coded symbols in $\frac{n}{r+1}$ non-overlapping $(r+1)$-groups. The algorithmic proof that we used, relied on the fact that collecting $(r+1)$-groups, is one of the ways to achieve the lower bound on the size of $\mathcal{S}$. This observation will lead us to an achievability proof for the case of $(r+1)|n$. We will see that pair-wise disjoint repair groups is one of the (possibly many) arrangements of repair groups that leads to optimal constructions.*

Remark 7. *In the above bound, if we set $\alpha = 1$ and $M = k$, we get the same bound as [44]. The $\alpha = 1$ case is equivalent to considering scalar codes. As it turns out, for the scalar case, linear codes are sufficient for this bound and nonlinearity in the encoding process does not come with any improvements in code distance.*

In the following section, we show that the above distance bound is tight when $(r+1)|n$. This does not rule out that the bound is tight under more general assumptions, however, this is left as an open question. For linear codes, [44] shows that codes with information-symbol locality can be constructed under different assumptions (for example when $r|k$ and $2 < d < r+3$), using a structure theorem (*e.g.*, see Theorem 15 in [44]). At the same time, it is impossible to construct optimal and linear LRCs (with all-symbol locality) when $2 < d < r+3$ and $r|k$ (*e.g.*, see Corollary 10 in [44]). It would be interesting to explore the use of the tools presented in [44], to provide further impossibility, or achievability results that extend the $(r+1)|n$ case that we study.

4.4 Achievability of the Bound: Random LRCs

In this section, we establish the following existence result:

Theorem 3. *Let $(r+1)|n$ and $r \leq n-d$. Then, there exist (n, r, d, M, α)-LRCs with minimum distance*

$$d = n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{ra} \right\rceil + 2,$$

over a sufficiently large finite field.

We establish the theorem through capacity achieving schemes for a specific communication network; such network will be defined through a directed and acyclic flow-graph. In the first subsection, we introduce the communication model for our network. In the second subsection, we show that a capacity achieving scheme for the aforementioned network maps to specific codes with specified parameters. In the third and fourth subsections, we construct randomized capacity achieving schemes and then map them to (n, r, d, M, α)-LRCs. The distance d of the aforementioned codes will be equal to the upper bound of Theorem 1.

4.4.1 The flow-graph network, multicast sessions, and its multicast capacity

Our achievability proof relies on using random linear network coding (RLNC) on a directed acyclic flow-graph, borrowing ideas from [34, 47]. Fig. 4.2 shows the directed and acyclic flow-graph that we use, which is formally defined subsequently.

At a conceptual level our proof analyzes a nested multicast problem that consists of two parts. We show that when RLNC is employed on the flow-graph in Fig. 4.2, *i)* it multicasts the source transmitted by node X to all data collectors (global decoding requirements) and *ii)* it simultaneously allows each collection of r nodes Y_j^{out}, originating from the same Γ_i node, to reconstruct whatever Γ_i transmits (local decoding requirements). The first part of the proof is a standard application of RLNC [47]. For the second part, our proof relies on a further subtle technicality that we discuss below.

General nested multicasting problems can be very challenging, but our problem has a very special structure: there are no edges between Y_j^{in}, $Y_{j'}^{\text{out}}$ nodes that originate from different Γ_i vertices. This means that there is no "algebraic interference" between the linear combinations of packets transmitted/received by these nodes. We use this fact to show that if the T data collectors $\text{DC}_1, \ldots, \text{DC}_T$ receive linearly independent equations of the source information, then each group

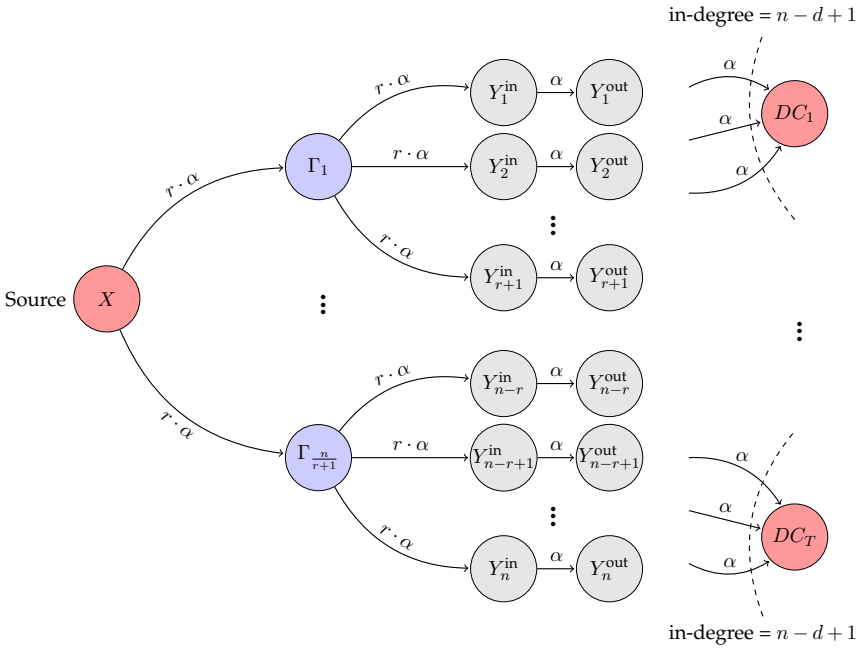


Figure 4.2: The directed acyclic information flow-graph $\mathcal{G}(n, r, d, \alpha)$. The left-most vertex is the source node X. The $\frac{n}{r+1}$ vertices Γ_i correspond to nodes that limit the in-flow to specific groups of intermediate nodes. The right-most $T = \binom{n}{n-d+1}$ vertices DC_i are the destination nodes (referred to as the data collectors) of the network. Each DC is connected to a different $(n-d+1)$-tuple of Y_i^{out} nodes.

of r nodes Y_j^{out} that originate from the same Γ_i node, receive linearly independent equations of the packets that node Γ_i transmits. This allows us to essentially use the technique of Ho *et al.* [47] to establish that both the global and local decoding requirements are simultaneously satisfied.

We now proceed with the detailed description of our proof. Let $\mathcal{G}(n, r, d, \alpha)$, be a directed acyclic graph that represents a communication network with 1 source node and T destination nodes and has vertex set

$$\mathcal{V} = \left\{ X, \Gamma_1, \ldots, \Gamma_{\frac{n}{r+1}}, Y_1^{\text{in}}, Y_1^{\text{out}}, \ldots, Y_n^{\text{in}}, Y_n^{\text{out}}, \text{DC}_1, \ldots, \text{DC}_T \right\},$$

where X denotes the *source node*, $\text{DC}_1, \text{DC}_2, \ldots, \text{DC}_T$ are the $T = \binom{n}{n-d+1}$ *destination*

nodes, referred to as the Data Collectors (DCs), and the remaining nodes are the *intermediate nodes*. Each vertex in $\mathcal{V}$ is assumed to be a receive and/or transmit *node*. It will become clear what that means after the following definitions, which are introduced to make our proof self-contained, while requiring minimal familiarity with network coding theory. For further details on our network model please refer to [106].

Definition 4 (edge capacity/network use/local encoding function/source message)**.** *A directed edge between two vertices v and u denotes a communication link between two nodes, over which bits are transmitted. All links are assumed to introduce no error. The directed edge capacity $c(v,u)$, between vertices v,u, denotes the maximum number of bits that can be communicated from node v to node u during a single network use. A single network use denotes the sequence of single transmissions over every directed edge. A message $\mathbf{m}_{(v,u)}$ is a collection of $c(v,u)$ bits that are transmitted from node v to node u, during a single network use. A message $\mathbf{m}_{(v,u)}$ can be considered as a collection of $c(v,u)$ binary uniform variables*[6] *with joint binary entropy equal to $H(\mathbf{m}_{(v,u)}) = c(v,u)$. Let $\mathcal{I}_v$ denote the (in-coming) vertices incident to vertex v. Then, the message $\mathbf{m}_{(v,u)}$ that is transmitted on a link $e(v,u)$ during a single network use, is the output of a local encoding function*

$$f_{v,u}\left(\left\{\mathbf{m}_{(v',v)}; v' \in \mathcal{I}_v\right\}\right) : \mathbb{F}_2^{\sum_{v'\in\mathcal{I}_v} c(v',v)} \to \mathbb{F}_2^{c(v,u)},$$

that takes as input the set of messages $\left\{\mathbf{m}_{(v',v)}; v' \in \mathcal{I}_v\right\}$ received by node v via the incident nodes in $\mathcal{I}_v$. The source node of the network holds a source bit sequence $\mathbf{x}$ of size $H(\mathbf{x})$ bits and wishes to transmit it to the T destination nodes.

We are now ready to define the directed weighted edge (link) set, that is determined by the following link capacities

$$c(v,u) = \begin{cases} r\cdot\alpha, & (v,u) = (X,\Gamma_j), \forall j \in \left[\frac{n}{r+1}\right], \\ r\cdot\alpha, & (v,u) = (\Gamma_j, Y_l^{\text{in}}), \forall j \in \left[\frac{n}{r+1}\right] \text{ and } l \in \{(j-1)(r+1)+1,\ldots,j(r+1)\}, \\ \alpha, & (v,u) = \left(Y_j^{\text{in}}, Y_j^{\text{out}}\right), \forall j \in [n], \\ \alpha, & (v,u) = (Y_j^{\text{out}}, \text{DC}_t), \forall j \in \mathcal{F}_t \text{ and } t \in [T], \\ 0, & \text{otherwise}, \end{cases}$$

[6] Although we assume that the messages transmitted over the links are sets of binary variables, this can be generalized to 2^τ-ary variables (*i.e.*, each variable will now be an element of a finite field of order $q = 2^\tau$). This is possible, if we consider τ consecutive transmit sessions per link, during a network use. We can then consider an equivalent network where the alphabet is 2^τ-ary. As a consequence, the entropies used under this setting should be 2^τ-ary, and all the following results carry on to that case.

where the T sets $\mathcal{F}_1, \ldots, \mathcal{F}_T$ are all $T = \binom{n}{n-d+1}$ possible subsets of $n-d+1$ integers in $[n]$. Observe that the in-degree of any DC_t node (the number of incident directed edges arriving at that node) is $n-d+1$.

The $\mathcal{G}(n, r, d, \alpha)$ network comes together with T *decoding requirements*: each destination node DC_t, for $t \in [T]$, is required, after a network use, to be able to reproduce from its received bits the source sequence $\mathbf{x}$. The decoding requirements can be stated as the following conditional entropy requirements:

$$\mathcal{D}_t: \quad H\left(\mathbf{x} \middle| \left\{\mathbf{m}_{(Y_j^{\text{out}}, DC_t)} : j \in \mathcal{F}_t\right\}\right) = 0, \ \forall t \in [T].$$

We are now ready to provide the main definition needed for our proof.

Definition 5 (multicast capacity and capacity achieving schemes)**.** *The directed graph $\mathcal{G}(n, r, d, \alpha)$ and a set of decoding requirements $\mathcal{D}_1, \ldots, \mathcal{D}_T$, specify a multicast connection problem. Let C be the maximum number of source bits such that all decoding requirements are satisfied, after a single network use. Then, C is defined as the multicast capacity of $\mathcal{G}(n, r, d, \alpha)$. A capacity achieving scheme, is a collection of local encoding functions such that all T decoding requirements are satisfied for $H(\mathbf{x}) = C$.*

In the following subsection, we derive a connection between a capacity achieving scheme on $\mathcal{G}(n, r, d, \alpha)$ and the existence of a code of well specified parameters. Then, we calculate the capacity of $\mathcal{G}(n, r, d, \alpha)$.

4.4.2 Connecting capacity achieving schemes to codes

The following lemma connects capacity achieving schemes on $\mathcal{G}(n, r, d, \alpha)$, to the existence of codes.

Lemma 5. *The set of n local encoding functions $f_{(Y_i^{in}, Y_i^{out})}$, $i \in [n]$, of a capacity achieving scheme on $\mathcal{G}(n, r, d, \alpha)$, can be mapped to a code of length n, that encodes a file of size C in n coded symbols, each of size α bits. This code has distance d.*

Proof. Observe that any local encoding function $f_{v,u}$ can be re-written as some *global encoding function* of the C source bits in $\mathbf{x}$ [106]. Let $f_i(\mathbf{x}) : \mathbb{F}_2^M \to \mathbb{F}_2^\alpha$ be the global function representation for $f_{(Y_i^{\text{in}}, Y_i^{\text{out}})}$. If the n local encoding functions

$$f_{(Y_1^{\text{in}}, Y_1^{\text{out}})}, \ldots, f_{(Y_n^{\text{in}}, Y_n^{\text{out}})}$$

are part of a capacity achieving scheme, then due to the decoding requirements being satisfied, we have

$$H\left(\mathbf{x}|\left\{\mathbf{m}_{(Y_j^{\text{out}},\text{DC}_t)} : j \in \mathcal{F}_t\right\}\right) = 0 \Rightarrow H\left(\mathbf{x}|\{f_j(\mathbf{x}) : j \in \mathcal{F}_t\}\right) = 0 \tag{4.18}$$

since $\mathbf{m}_{(Y_j^{\text{out}},\text{DC}_t)}$ is a function of $f_j(\mathbf{x})$, $t \in [T]$. Now, let $Y_i = f_i(\mathbf{x})$ and observe that each Y_i is a collection of α bits. Then, the T decoding requirements

$$H\left(\mathbf{x}|\{f_i(\mathbf{x}) : i \in \mathcal{F}_t\}\right) = 0,$$

for $t \in [T]$, are equivalent to the following statement: "any collection of $n-d+1$ symbols Y_i, with $i \in [n]$, are sufficient to reconstruct $\mathbf{x}$". This implies that

$$[Y_1, \dots Y_n] = [f_1(\mathbf{x}) \dots f_n(\mathbf{x})], \tag{4.19}$$

defines a code of length n, that encodes a files of size C, each coded symbol is of size α, and any $n-d+1$ coded symbols can reconstruct $\mathbf{x}$, *i.e.*, the code has distance d. □

Remark 8. *Observe that the above result does not guarantee that the code defined by $f_{(Y_i^{in},Y_i^{out})}$ has locality r. Locality comes as an artifact of the graph structure and the random capacity achieving scheme that we will use.*

4.4.3 Computing the source-destination cuts and achieving the capacity

In this subsection, we calculate the capacity of $\mathcal{G}(n,r,d,\alpha)$, and show how to achieve it. Let us first define the minimum cuts of the above network.

Definition 6 (minimum cut)**.** *A directed cut between nodes v and u, referred to as Cut$(v,u) \subseteq E$, is a subset of directed edges, such that if these edges are removed, then there does not exist a directed path between nodes v and u; $|$Cut$(v,u)|$ is the sum of all edge capacities in the set Cut(v,u), referred to as the capacity of Cut(v,u). A minimum cut MinCut(v,u) is the cut with the minimum aggregate edge capacity among all cuts between v and u.*

It is a well-known fact for communication networks, that $|$MinCut$(v,u)|$ is an upper bound on the number of bits that one can communicate from node v to node u [106]. Consequently, the cut with the minimum capacity, among the

cuts of all source-destination pairs, is an *upper bound* on the multicast capacity of a network. Most importantly, we know that this bound is achievable for multicast session networks. We state as Theorem 3, what is a collection of breakthrough results from [4,47].

Theorem 4 ([4,47]). *The multicast capacity C of a network with 1 source and T destination nodes, is equal to the minimum number among all capacities of minimum source-destination cuts. The capacity is achievable using random linear network coding.*

Remark 9. *In our case, RLNC stands for having local encoding functions*

$$f_{v,u}\left(\left\{\mathbf{m}_{(v',v)}; v' \in \mathcal{I}_v\right\}\right) : \mathbb{F}_q^{\sum_{v' \in \mathcal{I}_v} c(v',v)} \to \mathbb{F}_q^{c(v,u)},$$

for all u, v, such that the outputs of each of those functions are $c(v,u)$ symbols over a q-ary alphabet, and each output symbol is a linear combination of the $\sum_{v' \in \mathcal{I}_v} c(v',v)$ input symbols; each of these linear combinations has coefficients that are picked uniformly at random from the q-ary alphabet.

We use the above results and definitions to prove the key technical lemma of this subsection. Before we proceed with that, we present some properties of the ceiling and floor functions that are used in our proof.

Proposition 1 ([63]). *Let n and m be positive integer numbers, and x any real number. Then, the following hold*
(i) $\left\lfloor \frac{n}{m} \right\rfloor = \left\lceil \frac{n+1}{m} \right\rceil - 1$, *(ii)* $\left\lceil \frac{x+m}{n} \right\rceil = \left\lceil \frac{\lceil x \rceil + m}{n} \right\rceil$, *(iii)* $\left\lceil \frac{\lceil x/m \rceil}{n} \right\rceil = \left\lceil \frac{x}{nm} \right\rceil$.

We now proceed with the main lemma.

Lemma 6. *Let $d = n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{r \cdot \alpha} \right\rceil + 2$. Then, then the multicast capacity of the $\mathcal{G}(n,r,d,\alpha)$ network is equal to*

$$C = \left\lceil \frac{M}{\alpha} \right\rceil \alpha \geq M. \tag{4.20}$$

Proof. Using Theorem 3, the capacity of $\mathcal{G}(n,r,d,\alpha)$ is equal to

$$\min_{t \in [T]} \left| \text{MinCut}\left(X, \text{DC}_t\right) \right|.$$

Let us calculate the minimum cut capacity among all minimum cuts. Let us denote as the i-th $(r+1)$-group, the set of $r+1$ intermediate nodes Y_j^{out} that can be reached

from Γ_i. Now consider a DC that connects to a set of $n-d+1$ nodes *including* all the nodes of, say, the first $(r+1)$-group, and assume without loss of generality that this is DC_1. There are two (meaningful) choices for $\text{Cut}(X, \text{DC}_1)$: *i)* it can consist of all $(r+1)$ edges $(Y_i^{\text{in}}, Y_i^{\text{out}})$, $i \in [r+1]$, of the $(r+1)$-group, or *ii)* it can consist of simply the (X, Γ_1) edge.[7]

Clearly, the latter choice leads to a smaller cut capacity, since (X, Γ_1) has capacity $r \cdot \alpha$, whereas the $r+1$ edges $(Y_i^{\text{in}}, Y_i^{\text{out}})$, $i \in [r+1]$, have an aggregate capacity of $(r+1) \cdot \alpha$. Hence, for every cut that includes $r+1$ edges of the $(Y_j^{\text{in}}, Y_j^{\text{out}})$ kind that belong to the same $(r+1)$-group, say the i-th, then (X, Γ_i) can be used instead, reducing the capacity of such cut. Therefore, the smallest source-destination cut is the one that contains the largest possible number of (X, Γ_i) edges.

Now, the minimum aggregate capacity among all these T cuts, *i.e.*,

$$\min_{t=1,\ldots,T} |\text{MinCut}(X, \text{DC}_t)|,$$

will be the one that corresponds to the minimum cut of the DC that covers entirely as many $(r+1)$-groups as possible. Since the total number of Y_j^{out} nodes that a DC connects to is $n-d+1$, then the number of $(r+1)$-groups it can entirely cover is[8] $\left\lfloor \frac{n-d+1}{r+1} \right\rfloor$. The minimum cut will hence include

$$n_1 = \left\lfloor \frac{n-d+1}{r+1} \right\rfloor$$

edges of the (X, Γ_i) kind, which contribute to the cut an aggregate capacity of $n_1 r \alpha$. The remaining capacity comes from cutting a number of

$$n_2 = n-d+1-n_1 = n-d+1-(r+1)\left\lfloor \frac{n-d+1}{r+1} \right\rfloor$$

edges of the $(Y_i^{\text{in}}, Y_i^{\text{out}})$ kind. Therefore, we have that the smallest source-DC cut is

[7] The assumption $r \leq n-d$ is made such that $n-d+1 \geq r+1$. This implies that a DC has to connect to at least $r+1$ nodes.

[8] We would like to note here that the ratio inside the floor function is never an integer number: if it was, then all DCs could connect to exactly one less Y_i^{out} node while maintaining exactly the same source-destination cut.

equal to

$$\begin{aligned}
&\min_{t\in[T]} |\text{MinCut}\,(X,\text{DC}_t)| \\
&= n_1\cdot r\cdot\alpha + n_2\cdot\alpha = \left(n-d+1-\left\lfloor\frac{n-d+1}{r+1}\right\rfloor\right)\alpha \\
&= \left(\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{r\alpha}\right\rceil-1-\left\lfloor\frac{\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{r\alpha}\right\rceil-1}{r+1}\right\rfloor\right)\alpha \\
&\overset{(i)}{=} \left(\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{r\alpha}\right\rceil-1-\left\lceil\frac{\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{r\alpha}\right\rceil}{r+1}\right\rceil+1\right)\alpha \\
&\overset{(iii)}{=} \left(\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{r\alpha}\right\rceil-\left\lceil\frac{\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{\lceil M/\alpha\rceil}{r}\right\rceil}{r+1}\right\rceil\right)\alpha \\
&\overset{(ii)}{=} \left(\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{r\alpha}\right\rceil-\left\lceil\frac{\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{\alpha}\right\rceil\frac{1}{r}}{r+1}\right\rceil\right)\alpha = \left(\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{r\alpha}\right\rceil-\left\lceil\frac{\left\lceil\frac{M}{\alpha}\right\rceil\frac{r+1}{r}}{r+1}\right\rceil\right)\alpha \\
&= \left(\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{r\alpha}\right\rceil-\left\lceil\frac{\left\lceil\frac{M}{\alpha}\right\rceil}{r}\right\rceil\right)\alpha \overset{(iii)}{=} \left(\left\lceil\frac{M}{\alpha}\right\rceil+\left\lceil\frac{M}{r\alpha}\right\rceil-\left\lceil\frac{M}{r\alpha}\right\rceil\right)\alpha = \left\lceil\frac{M}{\alpha}\right\rceil\cdot\alpha \geq M, \quad (4.21)
\end{aligned}$$

on the above derivations we explicitly state which of the three properties of the ceiling/floor function found in Proposition 1 we are using. The above establishes our lemma. □

Using Lemma 2, Theorem 3, and Lemma 1, we obtain the following corollary.

Corollary 1. *There exists a capacity achieving scheme for $\mathcal{G}(n,r,d,\alpha)$, whose local encoding functions $f_{(Y_i^{in},Y_i^{out})}$, for $i\in[n]$, map to a code of length n that encodes $M^*\in\left[M,\left\lceil\frac{M}{\alpha}\right\rceil\alpha\right]$ source symbols in n coded symbols of size α, and the code has distance*[9] *$d=n-\left\lceil\frac{M}{\alpha}\right\rceil-\left\lceil\frac{M}{r\cdot\alpha}\right\rceil+2$.*

Observe that we are not done yet: we still have to prove that be above code has locality r. The next subsection finalizes our proof, by showing that RLNC on $\mathcal{G}(n,r,d,\alpha)$ indeed implies codes with locality r and distance matching our bounds, for a sufficiently large finite field.

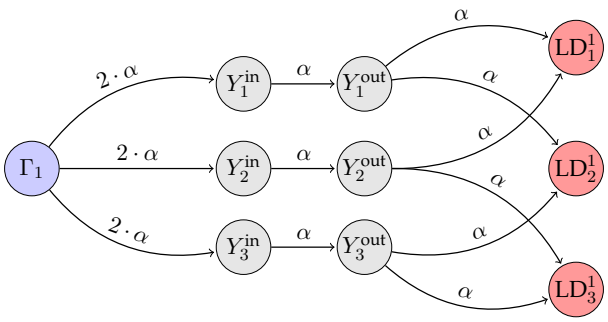


Figure 4.3: The $\mathcal{G}_1$ subgraph induced by the first $(r+1)$-group of a $\mathcal{G}(n, r=2, d, \alpha)$ network. The additional LD_j^i are local data collectors that are conceptual. These local DCs are not present in the original graph, and are used here to finalize the proof of Theorem 3. We use them to establish the locality of the code obtained through the RLNC capacity achieving scheme on $\mathcal{G}(n, r=2, d, \alpha)$.

4.4.4 Establishing the locality of the code and concluding the proof

To establish the locality of the code obtained in the previous subsection, we will show that an extra set of *local decoding requirements* are satisfied when RLNC is used. For this part of the proof we will focus on the subgraphs induced by the $(r+1)$-groups. Let $\mathcal{G}_i$, be the subgraph that is induced by the vertices

$$\mathcal{V}_i = \left\{\Gamma_i, Y^{\text{in}}_{(i-1)\cdot(r+1)+1}, Y^{\text{out}}_{(i-1)\cdot(r+1)+1}, \ldots, Y^{\text{in}}_{i\cdot(r+1)}, \ldots, Y^{\text{out}}_{i\cdot(r+1)}\right\},$$

for any $i \in \left[\frac{n}{r+1}\right]$. Let us assume that for each of these subgraphs there exists an additional number of $\binom{r+1}{r} = r+1$ *local Data Collector* nodes, $\text{LD}_1^i, \ldots, \text{LD}_{r+1}^i$. Each local DC is connected to one of the $r+1$ possible r-subsets of Y_j^{out} nodes of $\mathcal{G}_i$, with

$$j \in \{(i-1)(r+1)+1, \ldots, i(r+1)\}.$$

In Fig. 4.3, we give an example of $\mathcal{G}_1$ with the added local DCs.

[9]Observe that here we obtain the result for $M^* \in \left[M, \left\lceil \frac{M}{\alpha} \right\rceil \alpha\right]$. One can easily inspect that by substituting the M^* value in the distance bound, this value does indeed respect it.

Each of these local DCs has a decoding requirement: it requires to be able to decode what was transmitted by Γ_i. Let us refer to such a decoding requirement for the j-th local DC of $\mathcal{G}_i$ as $\mathcal{LD}_j^i$.

Remark 10. *Observe that the decoding requirement $\mathcal{LD}_j^i$ implies that the j-th local DC can reconstruct any single of the $r+1$ messages $\mathbf{m}_{(Y_j^{in}, Y_j^{out})}$, with $j \in \{(i-1)(r+1)+1, \ldots, i(r+1)\}$. This is true since all these $r+1$ messages are functions of what is transmitted by the Γ_i node.*

The above observation will be used to establish the locality of the codes obtained from RLNC on $\mathcal{G}(n, r, d, \alpha)$. Before we do that, we will state the following lemma, which will help us to conclude our proof.

Lemma 7. *When RLNC is used on $\mathcal{G}_i$, the decoding requirement $\mathcal{LD}_j^i$ is equivalent to a full-rank requirement $\mathcal{FR}_j^i$ on an $r \cdot \alpha \times r \cdot \alpha$ matrix with random i.i.d. coefficients.*

Proof. Without loss of generality, let us consider $\mathcal{G}_1$, moreover, let for simplicity

$$\mathbf{z}_1 \in \mathbb{F}_q^{1 \times r \cdot \alpha}$$

be the source message that Γ_1 wishes to transmit to the local DCs. Since the capacity of a $(\Gamma_1, Y_j^{\text{in}})$ edge is $r \cdot \alpha$, for $j \in [r+1]$, then node Y_j^{in} receives $\mathbf{z}_1$. Moreover, due to the RLNC scheme used, the coefficients of the random linear combinations in the local encoding functions are picked independently. Hence, node Y_j^{in} will transmit to node Y_j^{out} a vector of α symbols:

$$\mathbf{z}_1 \mathbf{A}_{1,j}$$

where $\mathbf{A}_{1,j}$ is an $r \cdot \alpha \times \alpha$ matrix of random i.i.d. coefficients. Then, any node Y_j^{out} transmits to the local DCs of $\mathcal{G}_1$ exactly what it received, *i.e.,* $\mathbf{z}_1 \mathbf{A}_{1,j}$, since the capacity of the edge $(Y_j^{\text{out}}, \text{LD}_i^1)$ is α. Hence, any local DC receives r vectors of size α, which if put together form a vector of size $r \cdot \alpha$; this vector, for local DC j, can be re-written as $\mathbf{z}_1 \mathbf{A}_j^1$, where $\mathbf{A}_j^1$ is an $r \cdot \alpha \times r \cdot \alpha$ matrix of random i.i.d. coefficients. Hence, any local DC decoding requirement is equivalent a requirement on a square matrix of random coefficients being full-rank. Let us refer to this full-rank requirement as $\mathcal{FR}_j^i$. □

Observe that $\mathcal{FR}_j^i$ is a requirement that can be stated independently of the existence of local DCs. Hence, we can now go back on $\mathcal{G}(n, r, d, \alpha)$ and show that RLNC allows all local decoding requirements and all $\mathcal{FR}_j^i$ conditions to be satisfied *simultaneously*.

Lemma 8. *Let*

$$d = n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{ra} \right\rceil + 2$$

and let us employ RLNC on $\mathcal{G}(n, r, d, \alpha)$. Then, all decoding requirements $\mathcal{D}_i$ of $\mathcal{G}(n, r, d, \alpha)$ and all full rank requirements $\mathcal{FR}_j^i$ will be simultaneously satisfied, with nonzero probability, when the finite field is sufficiently large.

Proof. Let $E_{\mathcal{G}}$ denote the event that some of the T DCs of $\mathcal{G}(n, r, d, \alpha)$ cannot decode $\mathbf{x}$ successfully, which say, has probability p_1 that is a function of the size of the finite field used by the RLNC scheme [47]. Moreover, let $E_{\mathcal{FR}_j^i}$ denote the event that $\mathcal{FR}_j^i$ is not satisfied, which say, has probability p_2 that is also a function of the size of the finite field used by the RLNC scheme. Then, the probability that RLNC does not satisfy some of the above conditions is

$$\Pr\left\{ E_{\mathcal{G}} \bigcup \left\{ \bigcup_{i=1}^{n/(r+1)} \bigcup_{j=(i-1)(r+1)+1}^{i(r+1)} E_{\mathcal{FR}_j^i} \right\} \right\}$$
$$\leq \Pr\{E_{\mathcal{G}}\} + \sum_{i=1}^{n/(r+1)} \sum_{j=(i-1)(r+1)+1}^{i(r+1)} \Pr\left\{ E_{\mathcal{FR}_j^i} \right\} = p_1 + n \cdot p_2.$$

We can now conclude our proof, since p_1 and p_2 can be made arbitrarily small, using a sufficiently large finite field [47]. □

Due to the above lemma and Lemma 1, we use the $f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})$ global encoding functions (the global representations of the $f_{(Y_i^{\text{in}}, Y_i^{\text{out}})}$s) of the RLNC scheme to obtain a code that encodes a file of size M in n coded symbols, each of size α; such code has distance d.

Moreover, since all $\mathcal{FR}_j^i$ requirements are satisfied, then as mentioned in Remark 10, each output of a global encoding function $f_i(\mathbf{x})$ can be reconstructed from the outputs of a subset of r other global encoding functions: this implies

repair locality r. Hence, the code defined by the global encoding functions f_i is an (n, r, d, M, α)-LRC, with

$$d = n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{ra} \right\rceil + 2.$$

This concludes the proof of Theorem 2.

4.5 Locally Repairable Codes: Explicit Constructions

In this section, we provide an explicit LRC family for the operational point on the distance trade-off, where any k subsets of coded nodes can reconstruct all k file symbols, *i.e.*, when $d = n - k + 1$. For this regime, that resembles that of an (n, k)-MDS code, we will show how to achieve the distance of an (n, k)-MDS code, while having locality $r << n$ and sacrificing only a small fraction of the code rate: the rate of our codes will be $\frac{1}{r}\frac{k}{n}$ less than that of an (n, k)-MDS code. Specifically, the code parameters for our LRCs are

$$\left(n, r, d = n - k + 1, M, \alpha = \frac{r+1}{r} \cdot \frac{M}{k}\right), \text{ such that } (r+1)|n.$$

Our codes meet the optimal distance bound for all of the above coding parameters when $(r + 1) \nmid k$.

The presented codes come with the following design advantages: *i)* they achieve arbitrarily high data rates, *ii)* they can be constructed using Reed-Solomon encoded blocks, *iii)* the repair of a lost node requires downloading blocks and XORing them at a destination node, and *iv)* their vector size, or sub-packetization length, is r, and each stored sub-symbol is over a small finite field with size proportional to n. This means that we can represent each coded symbol by using only $r \cdot O(\log n)$ bits.

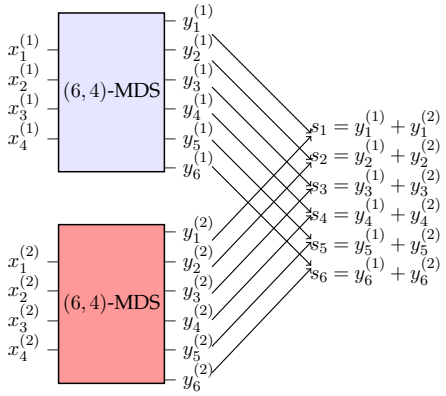
4.5.1 Code construction

Let a file $\mathbf{x}$ of size $M = rk$ symbols[10], that is sub-packetized in r parts,

$$\mathbf{x} = \left[\mathbf{x}^{(1)} \dots \mathbf{x}^{(r)}\right],$$

[10]here the size of each symbol depends on the code construction, and is not necessarily binary. As we see in the following, the size of each symbol will be proportional to $\log(n)$ bits.

MDS Pre-coding and XORing



with each $\mathbf{x}^{(i)}$, $i \in [r]$, having size k. We encode each of the r file parts independently, into coded vectors $\mathbf{y}^{(i)}$ of length n, where $(r+1)|n$, using an outer (n, k) MDS code

$$\mathbf{y}^{(1)} = \mathbf{x}^{(1)}\mathbf{G}, \ \ldots, \ \mathbf{y}^{(r)} = \mathbf{x}^{(r)}\mathbf{G},$$

where $\mathbf{G}$ is an $n \times k$ MDS generator matrix.

As MDS pre-codes, we use (n, k)-RS codes that require each of the k elements to be over a finite field $\mathbb{F}_{2^p}$, for any p such that $2^p \geq n$. This will imply that all stored sub-symbols in our code are over a finite field of size $2^p \geq n$. We then generate a single parity XOR vector from all the coded vectors

$$\mathbf{s} = \bigoplus_{i=1}^{r} \mathbf{y}^{(i)}.$$

The above *pre-coding* process yields a total of $r \cdot n$ coded blocks, the $\mathbf{y}^{(i)}$ vectors and n XOR parity blocks in the $\mathbf{s}$ vector. That is, we have an aggregate of $(r+1)n$ blocks available to place in n nodes, hence we decide to store $r+1$ blocks per node. Therefore, each node needs to have a storage capacity of

$$\alpha = \frac{M}{k} + \frac{1}{r}\frac{M}{k} = r + 1 \text{ (coded blocks)}.$$

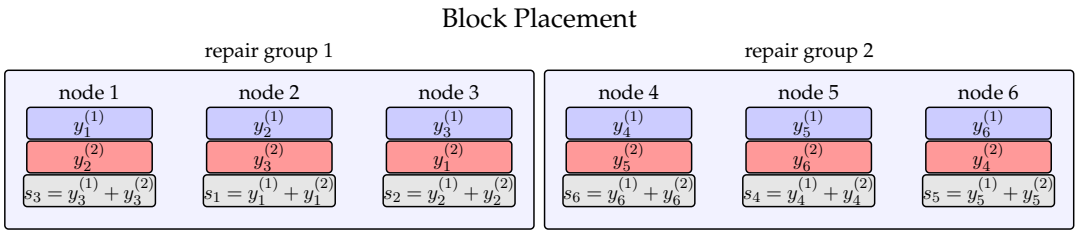


Figure 4.4: MDS pre-coding, XORing, and block placement in nodes.

	node 1	node 2	...	node r	node $r+1$
blocks of $\mathbf{y}^{(1)}$	$y_1^{(1)}$	$y_2^{(1)}$	...	$y_r^{(1)}$	$y_{r+1}^{(1)}$
blocks of $\mathbf{y}^{(2)}$	$y_2^{(2)}$	$y_3^{(2)}$	...	$y_{r+1}^{(2)}$	$y_1^{(2)}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
blocks of $\mathbf{y}^{(r)}$	$y_r^{(r)}$	$y_{r+1}^{(r)}$	...	$y_{r-2}^{(r)}$	$y_{r-1}^{(r)}$
blocks of $\mathbf{s}$	s_{r+1}	s_1	...	s_{r-1}	s_r

Table 4.1: The first $r+1$ nodes in our code construction. These nodes belong to the first $(r+1)$-repair group. The nodes in the remaining repair groups have a block placement that follows the same circular-shifting pattern.

In Table 4.1, we state the circular placement of symbols in nodes of the first $(r+1)$-group . There are three key properties of the block placement:

1. each node contains r coded blocks coming from different $\mathbf{y}^{(l)}$ coded vectors and 1 additional parity symbol,

2. the blocks in the $r+1$ nodes of the i-th $(r+1)$-group have indices that appear only in that specific repair group, and

3. the blocks of each row have indices that obey a circular pattern, *i.e.*, the first row of symbols has index ordering $\{1, 2, \ldots, r+1\}$, the second has ordering $\{2, 3, \ldots, r+1, 1\}$, and so on.

In Fig. 4.4, we show an LRC of the above construction with $M = 8$, $\alpha = 3$, $n = 6$ and $k = 4$, that has locality 2.

4.5.2 Repairing lost nodes

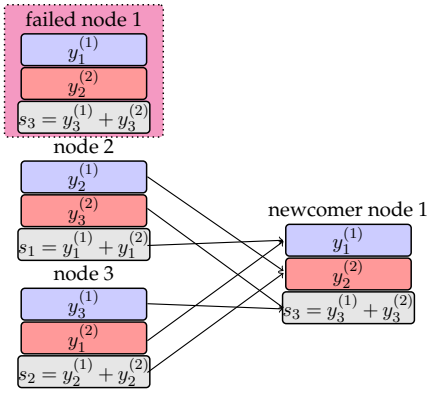


Figure 4.5: We show an example of a failed node repair. The repair locality here is 2 since 2 remaining nodes are involved in reconstructing the lost information of the first node. Observe that we repair a failed node by simply transferring blocks: no block combinations are need to be performed at the sender nodes. Once the blocks are transferred to a newcomer, a simple XOR suffices for reconstruction.

Here, we see that the repair of each lost node requires contacting r nodes, *i.e.*, the locality of the code is r. Without loss of generality, we consider the repair of a node in the first repair group of $r + 1$ nodes. This is sufficient since the nodes across different repair groups follow the same placement properties.

The key observation is that each node within a repair group stores $r + 1$ blocks of *distinct* indices: the $r + 1$ blocks of a particular index are stored in $r + 1$ distinct nodes within a single repair group. When for example the first node fails, then $y_1^{(1)}$, the symbol of the first row, is regenerated by downloading s_1 from the second node, $y_1^{(r+1)}$ from the third, and so on. Once all these symbols are downloaded, a simple XOR of all of them is exactly equal to $y_1^{(1)}$. In the same manner, for each node, in each repair group when we need to reconstruct a lost block, we first download the r remaining blocks of the same index and XOR them together

to regenerate the desired lost block. Since each block can be reconstructed by contacting r other blocks, and since the repair is confined within a single repair group of r remaining nodes, the code has locality r.

In Fig. 4.5, we show how repair is performed for the code construction presented in Fig. 4.4.

4.5.3 Distance and code rate

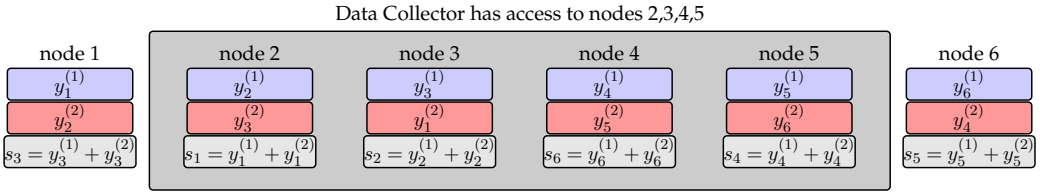


Figure 4.6: We show how the file can be reconstructed by contacting $k = 4$ nodes. Observe that by accessing *any* k nodes, a DC has access to k blocks from the first MDS code and k blocks from the second. Since the pre-codes are (n, k)-MDS, this means that any k blocks from each of the two coded blocks suffice to reconstruct both file parts.

The distance of the presented code is $d = n - k + 1$ due to the MDS pre-codes that are used in its design: any k nodes in the system contain rk distinct coded blocks, k from each of the r file blocks. Hence, by performing erasure decoding on each of these r k-tuples of blocks, we can generate the r blocks of the file.

When $(r + 1) \nmid k$ this distance matches the optimal bound,

$$n - \left\lceil \frac{M}{\alpha} \right\rceil - \left\lceil \frac{M}{r\alpha} \right\rceil + 2 = n - \left\lceil \frac{kr}{r+1} \right\rceil - \left\lceil \frac{k}{r+1} \right\rceil + 2 = n - k + 1,$$

since

$$\lceil rk/(r+1) \rceil + \lceil k/(r+1) \rceil = k + \lceil -k/(r+1) \rceil + \lceil k/(r+1) \rceil = k + 1,$$

when $k/(r + 1)$ is not an integer [63]. In Fig. 4.6, we give a file reconstruction example for the code of Fig. 4.4.

Finally, the effective coding rate of our LRC is

$$R = \frac{\text{size of useful information}}{\text{total storage spent}} = \frac{M}{n \cdot \alpha} = \frac{r}{r+1}\frac{k}{n}.$$

That is, the rate of the code is a fraction $\frac{r}{r+1}$ of the coding rate of an (n, k) MDS code, hence is always upper bounded by $\frac{r}{r+1}$. This loss in rate is incurred due to the use of the extra XOR stripe of blocks, that is required for efficient and local repairs. Observe that if we set the repair locality to $r = f(k)$ and f is a sub-linear function of k (*i.e.*, $\log(k)$ or $\sqrt{k}$), then we obtain non-trivially low locality $r << k$, while the excess storage cost $\epsilon = \frac{1}{r}$ is vanishing when n, k grow.

4.6 Conclusions

In this chapter, we presented *locally repairable codes*, a new family of repair efficient codes that optimize the metric of locality. We analyze what is the best possible reliability in terms of code distance, given the requirement that each coded symbol can be reconstructed by r other symbols in the code. We provide an information theoretic bound that ties together the code distance, the locality, and the storage cost of a code.

We prove that the above bound is achievable using vector-linear codes. Eventually, we give an explicit construction of LRCs for the case where we require that any k nodes can recover the encoded file. We show how this explicit construction not only has optimal locality, but also requires small field size and admits very simple XOR based repairs.

Part II

Large-Scale Data Processing

Chapter 5

Sparse PCA through Low-rank Approximations

In this chapter, we introduce a novel algorithm that approximates the k-sparse principal component of any arbitrary positive semidefinite matrix A. Our algorithm is combinatorial and operates by examining a discrete set of special vectors lying in a low-dimensional eigen-subspace of A. Our algorithm uses as a subroutine the constant-rank solver framework for sparse PCA that was recently developed by [10]. We obtain provable approximation guarantees that depend on the spectral profile of the matrix: the faster the eigenvalue decay, the better the quality of our approximation. For example, if the eigenvalues of A follow a power-law decay, we obtain a polynomial-time approximation algorithm for any desired accuracy.

A key algorithmic component of our scheme is a combinatorial feature elimination step that is provably safe and in practice significantly reduces the running complexity of our algorithm. We implement our algorithm and test it on multiple artificial and real data sets. Due to the feature elimination step, it is possible to perform sparse PCA on data sets consisting of millions of entries in a few minutes. Our experimental evaluation shows that our scheme is nearly optimal while finding very sparse vectors. We compare to the prior state of the art and show that our scheme matches or outperforms previous algorithms in all tested data sets. [1]

[1]Contributions Statement: Most parts of this chapter appear in [79]. Prof. Alex Dimakis supervised the project and all coauthors had equal contribution to this work. The constant rank solver used by the low-rank framework here, was introduced in [10]. The constant rank algorithm of [10] was not part of this thesis; the proof of its correctness is however included in Appendix Subsection B.1 for completeness .

5.1 Introduction

Principal component analysis (PCA) reduces the dimensionality of a data set by projecting it onto principal subspaces spanned by the leading eigenvectors of the sample covariance matrix. The statistical significance of PCA partially lies in the fact that the principal components capture the largest possible data variance. The first principal component (i.e., the first eigenvector) of an $n \times n$ matrix A is the solution to

$$\arg\max_{\|x\|_2=1} x^T A x$$

where $A = SS^T$ and S is the $n \times m$ data-set matrix consisting of m data-points, or entries, each evaluated on n features, and $\|x\|_2$ is the ℓ_2-norm of x. PCA can be efficiently computed using the singular value decomposition (SVD). The statistical properties and computational tractability of PCA renders it one of the most used tools in data analysis and clustering applications.

A drawback of PCA is that the generated vectors typically have very few zero entries, i.e., they are not *sparse*. Sparsity is desirable when we aim for *interpretability* in the analysis of principal components. An example where sparsity implies interpretability is document analysis, where principal components can be used to cluster documents and detect trends. When the principal components are sparse, they can be easily mapped to topics (e.g., newspaper article classification into politics, sports, etc.) using the few keywords in their support [41,109]. For that reason it is desirable to find sparse eigenvectors.

5.1.1 Sparse PCA

Sparsity can be directly enforced in the principal components. The sparse principal component x_* is defined as

$$x_* = \arg\max_{\|x\|_2=1, \|x\|_0=k} x^T A x. \tag{5.1}$$

The ℓ_0 cardinality constraint limits the optimization over vectors with k non-zero entries. As expected, sparsity comes at a cost since the optimization in (5.1) is NP-hard [72] and hence computationally intractable in general.

5.1.2 Overview of main results

We introduce a novel algorithm for sparse PCA that has a provable approximation guarantee. Our algorithm generates a k-sparse, unit length vector x_d that gives an objective provably within a $1 - \epsilon_d$ factor from the optimal:

$$x_d^T A x_d \geq (1 - \epsilon_d)\, x_*^T A x_*$$

with

$$\epsilon_d \leq \min \left\{ \frac{n}{k} \cdot \frac{\lambda_{d+1}}{\lambda_1},\ \ \frac{\lambda_{d+1}}{\lambda_1^{(1)}} \right\}, \tag{5.2}$$

where λ_i is the ith largest eigenvalue of A and $\lambda_1^{(1)}$ is the maximum diagonal element of A. For any desired value of the parameter d, our algorithm runs in time $O(n^{d+1} \log n + \text{SVD}(A, d))$, where $\text{SVD}(A, d)$ is the time to compute the d principal eigenvectors of A. Our approximation guarantee is directly related to the spectrum of A: the greater the eigenvalue decay, the better the approximation. Equation (5.2) contains two bounds: one that uses the largest eigenvalue λ_1 and one that uses the largest diagonal element of A, $\lambda_1^{(1)}$. Either bound can be tighter, depending on the structure of the A matrix.

We subsequently rely on our approximation result to establish guarantees for considerably general families of matrices.

5.1.2.1 Constant-factor approximation

If we only assume that there is an arbitrary decay in the eigenvalues of A, *i.e.*, there exists a constant $d = O(1)$ such that $\lambda_1 > \lambda_{d+1}$, then we can obtain a constant-factor approximation guarantee for the linear sparsity regime. Specifically, we find a constant δ_0 such that for all sparsity levels $k > \delta_0 n$ we obtain a constant approximation ratio for sparse PCA, partially solving the open problem discussed in [29, 108]. This result easily follows from our main theorem.

5.1.2.2 PTAS under a power-law decay

When the data matrix spectrum exhibits a power-law decay, we can obtain a much stronger performance guarantee: we can solve sparse PCA for any desired accuracy ϵ in time polynomial in n, k (but not in $\frac{1}{\epsilon}$). This is sometimes called a

polynomial-time approximation scheme (PTAS). Further, the power-law decay is not necessary: the spectrum does not have to follow exactly that decay, but only exhibit a substantial spectral drop after a few eigenvalues.

5.1.2.3 Algorithmic details

Our algorithm operates by scanning a low-dimensional subspace of A. It does so through a hyperspherical transformation on our problem space to reduce its dimensionality. This framework was introduced by [60] in the context of solving quadratic form maximization problems over ± 1 vectors. This framework was consequently used in [10] to develop a constant rank solver that computes the sparse principal component of a constant rank matrix in polynomial time $O(n^{d+1})$. We use the framework of [10] to examine a polynomial number of special vectors, that lead to a sparse principal component which admits provable performance. We tweak the solver of [10] to improve computation time by a factor of 2^d for matrices with nonnegative entries.

Although the complexity of our algorithm is polynomial in n, the cost to run it on even moderately large sets with $n > 1000$ becomes intractable even for small values of $d = 2$. A key algorithmic innovation that we introduce is a provably safe feature elimination step that allows the scalability of our algorithm for data-sets with millions of entries. We introduce a test that discards features that are provably not in the support of the sparse PC, in a similar manner as [109], but using a different combinatorial criterion.

5.1.2.4 Experimental Evaluation

We evaluate and compare our algorithm against state of the art sparse PCA approaches on synthetic and real data sets. Our real data-set is a large Twitter collection of more than 10 million tweets spanning approximately six months. We executed several experiments on various subsets of our data set: collections of tweets during a specific time-window, tweets that contained a specific word, etc. Our implementation executes in less than one second for $50k - 100k$ documents and in a few minutes for millions of documents, on a personal computer. Our scheme typically comes closer than 90% of the optimal performance, even for $d < 3$, and empirically outperforms previously proposed sparse PCA algorithms.

5.1.3 Related Work

There has been a substantial volume of prior work on sparse PCA. Initial heuristic approaches used factor rotation techniques and thresholding of eigenvectors to obtain sparsity [22, 55, 58]. Then, a modified PCA technique based on the LASSO (SCoTLASS) was introduced in [56]. In [110], a nonconvex regression-type approximation, penalized à la LASSO was used to produce sparse PCs. A nonconvex technique was presented in [93]. In [73], the authors used spectral arguments to motivate a greedy branch-and-bound approach, further explored in [74]. In [91], a similar technique to SVD was used employing sparsity penalties on each round of projections. A significant body of work based on semidefinite programming (SDP) approaches was established in [28, 30, 108]. A variation of the power method was used in [57]. When computing multiple PCs, the issue of deflation arises as discussed in [67]. In [107], the authors introduced a very efficient sparse PCA approximation based on truncating the well-known power method to obtain the exact level of sparsity desired. The constant rank solver of [10] that we use in our approximation algorithm, solves sparse PCA exactly for constant rank matrices in polynomial time. A fast algorithm based on Rayleigh quotient iteration was developed in [64].

Several guarantees are established under the statistical model of the spiked covariance. In [7], the first theoretical optimality guarantees were established under the spiked covariance for diagonal thresholding and the SDP relaxation of [30]. In [107], the authors provide peformance guarantees for the truncated power method under specific assumptions of data model, similar to the restricted isometry property. In [29] the authors provide detection guarantees under the single spike covariance model. Then, in [23] and [24] the authors provide guarantees under the assumption of multiple spikes in the covariance.

There has also been a significant effort in understanding the hardness of the problem. Sparse PCA is NP-hard in the general case as it can be recast to the problem subset selection and the problem of finding the largest clique in a graph. It is also suspected to be hard to recover the sparse spikes of a covariance under optimal sample complexity as was shown in [14], [13], and [12]. There, the problem of recovering the correct spike under the minimum possible sample complexity is connected to the problem of recovering a planted clique below the $\Theta(\sqrt{n})$ barrier.

In [10], the problem of sparse PCA is studied and a polynomial time solver is developed for matrices A that have constant rank d that is not a function of n. We use parts of that theoretical framework in our algorithm. The main algorithmic difference is that we speed up calculations by 2^d for matrices that have nonnegative entries. The key idea behind the constant-rank solver is a low-rank spherical transformation of the problem using a low-dimensional auxiliary vector. This machinery was introduced in the foundational work of [60].

Despite this extensive literature, to the best of our knowledge, there are very few provable approximation guarantees for sparse PCA algorithms. Usually such guarantees can be found when specific statistical data models are considered [7, 24, 29, 107].

5.2 Sparse PCA through Low-rank Approximations

5.2.1 Proposed Algorithm

Our algorithm is technically involved and for that reason we start with a high-level informal description. For any given accuracy parameter d we follow the following steps:

Step 1: *Obtain A_d, a rank-d approximation of A.*
We obtain A_d, the best-fit rank-d approximation of A, by keeping the first d terms of its eigen-decomposition:

$$A_d = \sum_{i=1}^{d} \lambda_i v_i v_i^T,$$

where λ_i is the i-th largest eigenvalue of A and v_i the corresponding eigenvector.

Step 2: *Use A_d to obtain $O(n^d)$ candidate supports.*
For any matrix A, we can exhaustively search for the optimal x_* by checking all $\binom{n}{k}$ possible $k \times k$ submatrices of A: x_* is the k-sparse vector with the same support as the submatrix of A with the maximum largest eigenvalue. However, we show how sparse PCA can be efficiently solved on A_d if the rank d is constant with respect to n, using the algorithm of [10]. The key technical fact proven there is that there are *only* $O(n^d)$ candidate supports that need to be examined. That is, a *set of candidate supports* $\mathcal{S}_d = \{\mathcal{I}_1, \ldots, \mathcal{I}_T\}$, where $\mathcal{I}_t$ is a subset of k indices from $\{1, \ldots, n\}$, contains

the optimal support. The number of these supports is[2]

$$|\mathcal{S}_d| \leq 2^{2d}\binom{n}{d}.$$

The above set $\mathcal{S}_d$ is efficiently created by the *Spannogram algorithm* described in the next subsection.

Step 3: *Check each candidate support from $\mathcal{S}_d$ on A.*
For a given support $\mathcal{I}$ it is easy to find the best vector supported on $\mathcal{I}$: it is the leading eigenvector of the principal submatrix of A, with rows and columns indexed by $\mathcal{I}$. In this step, we check all the supports in $\mathcal{S}_d$ *on the original matrix A* and output the best. Specifically, define $A_\mathcal{I}$ to be the zeroed-out version of A, except on the support $\mathcal{I}$. That is, $A_\mathcal{I}$ is an $n \times n$ matrix with zeros everywhere except for the principal submatrix indexed by $\mathcal{I}$. If $i \in \mathcal{I}$ and $j \in \mathcal{I}$, then $A_\mathcal{I} = A_{ij}$, else it is 0. Then, for any $A_\mathcal{I}$ matrix, with $\mathcal{I} \in \mathcal{S}_d$, we compute its largest eigenvalue and corresponding eigenvector.

Output:
Finally, we output the k-sparse vector x_d that is the principal eigenvector of the $A_\mathcal{I}$ matrix, $\mathcal{I} \in \mathcal{S}_d$, with the largest maximum eigenvalue. We refer to this approximate sparse PC solution as the *rank-d optimal solution*.

The exact steps of our algorithm are given in the pseudocode tables denoted as Algorithm 1 and 2. The spannogram subroutine, i.e., Algorithm 2, computes the T candidate supports in $\mathcal{S}_d$, and is presented and explained in Section 5.3. The complexity of our algorithm is equal to calculating d leading eigenvectors of A ($\mathcal{O}(SVD(A,d))$), running the spannogram algorithm ($\mathcal{O}(n^{d+1}\log n)$), and finding the leading eigenvector of $O(n^d)$ matrices of size $k \times k$ ($\mathcal{O}(n^d k^3)$). Hence, the total complexity is $O(n^{d+1}\log n + n^d k^3 + SVD(A,d))$.

Elimination Step: This step is run before Step 2. By using a feature elimination subroutine we can identify that certain variables provably cannot be in the support of x_d, the rank-d optimal sparse PC. We have a test which is related to the norms of the rows of V_d that identifies which of the n rows cannot be in the

[2] In fact, in the proof we show a better dependency on d, which however has a more complicated expression.

Algorithm 1 Sparse PCA via a rank-d approximation

Input: k, d, A
$p \leftarrow 1$ if A has nonnegative entries, 0 if mixed
$A_d \leftarrow \sum_{i=1}^{d} \lambda_i v_i v_i^T$
$\hat{A}_d \leftarrow$ feature_elimination(k, p, A_d)
$\mathcal{S}_d \leftarrow$ Spannogram$\left(k, p, \hat{A}_d\right)$
for each $\mathcal{I} \in \mathcal{S}_d$ **do**
    Calculate $\lambda_1(A_\mathcal{I})$
end for
$\mathcal{I}_d^{\text{opt}} = \arg\max_{I \in \mathcal{S}_d} \lambda_1(A_\mathcal{I})$
$\text{OPT}_d = \lambda_1(A_{I_d^{\text{opt}}})$
$x_d^{\text{opt}} \leftarrow$ the principal eigenvector of $A_{\mathcal{I}_d^{\text{opt}}}$.
Output: x_d^{opt}

optimal support. We use this step to further reduce the number of candidate supports $|\mathcal{S}_d|$. The elimination algorithm is very important when it comes to large scale data sets. Without the elimination step, even the rank-2 version of the algorithm becomes intractable for $n > 10^4$. However, after running the subroutine we empirically observe that even for n that is in the orders of 10^6 the elimination strips down the number of features to only around $50 - 100$ for values of k around 10. This subroutine is presented in detail in Appendix A.

5.2.2 Approximation Guarantees

The desired sparse PC is

$$x_* = \underset{\|x\|_2=1, \|x\|_0=k}{\arg\max} \; x^T A x.$$

We instead obtain the k-sparse, unit length vector x_d which gives an objective

$$x_d^T A x_d = \max_{\mathcal{I} \in \mathcal{S}_d} \lambda(A_\mathcal{I}).$$

We measure the quality of our approximation using the standard approximation factor:

$$\rho_d = \frac{x_d^T A x_d}{x_*^T A x_*} = \frac{\max_{\mathcal{I} \in \mathcal{S}_d} \lambda(A_\mathcal{I})}{\lambda_1^{(k)}},$$

where $\lambda_1^{(k)} = x_*^T A x_*$ is the k-sparse largest eigenvalue of A.[3] Clearly, $\rho_d \leq 1$ and as it approaches 1, the approximation becomes tighter. Our main result follows:

Theorem 5. *For any d, our algorithm outputs x_d, where $||x_d||_0$=k, $||x_d||_2$=1 and*

$$x_d^T A x_d \geq (1-\epsilon)\, x_*^T A x_*,$$

with an error bound

$$\epsilon_d \leq \frac{\lambda_{d+1}}{\lambda_1^{(k)}} \leq \min\left\{\frac{n}{k}\frac{\lambda_{d+1}}{\lambda_1}, \frac{\lambda_{d+1}}{\lambda_1^{(1)}}\right\}.$$

Proof. The proof can be found in Appendix 5. The main idea is that we obtain *i*) an upper bound on the performance loss using A_d instead of A and *ii)* a lower bound for $\lambda_1^{(k)}$. □

We now use our main theorem to provide the following model specific approximation results.

Corollary 2. *Assume that for some constant value d, there is an eigenvalue decay $\lambda_1 > \lambda_{d+1}$ in A. Then there exists a constant δ_0 such that for all sparsity levels $k > \delta_0 n$ we obtain a constant approximation ratio.*

Corollary 3. *Assume that the first $d+1$ eigenvalues of A follow a power-law decay, i.e., $\lambda_i = Ci^{-\alpha}$, for some $C, \alpha > 0$. Then, for any $k = \delta n$ and any $\epsilon > 0$ we can get a $(1-\epsilon)$-approximate solution x_d in time $O\left(n^{1/(\epsilon\delta)^\alpha + 1} \log n\right)$.*

The above corollaries can be established by plugging in the values for λ_i in the error bound. We find the above families of matrices interesting, because in practical data sets (like the ones we tested), we observe a significant decay in the first eigenvalues of A which in many cases follows a power law. The main point of the above approximability result is that any matrix with decent decay in the spectrum endows a good sparse PCA approximation.

[3] Notice that the k-sparse largest eigenvalue of A for $k = 1$ denoted by $\lambda_1^{(1)}$ is simply the largest element on the diagonal of A.

5.3 The Spannogram Algorithm

In this section, we describe how the Spannogram algorithm constructs the candidate supports in $\mathcal{S}_d$ and explain why this set has tractable size. We build up to the general algorithm by explaining special cases that are easier to understand.

5.3.1 Rank-1 case

Let us start with the rank 1 case, i.e., when $d = 1$. For this case

$$A_1 = \lambda_1 v_1 v_1^T.$$

Assume, for now, that all the eigenvector entries are unique. This simplifies tie-breaking issues that are formally addressed by a perturbation lemma in Appendix 5. For the rank-1 matrix A_1, a simple thresholding procedure solves sparse PCA: simply keep the k largest entries of the eigenvector v_1. Hence, in this simple case $\mathcal{S}_1$ consists of only 1 set.

To show this, we can rewrite (5.1) as

$$\max_{x \in \mathbb{S}_k} x^T A_1 x = \lambda_1 \cdot \max_{x \in \mathbb{S}_k} \left(v_1^T x\right)^2 = \lambda_1 \cdot \max_{x \in \mathbb{S}_k} \left(\sum_{i=1}^{n} v_{1i} x_i \right)^2, \tag{5.3}$$

where $\mathbb{S}_k$ is the set of all vectors $x \in \mathbb{R}^n$ with $||x||_2 = 1$ and $||x||_0 = k$. We are trying to find a k-sparse vector x that maximizes the inner product with a given vector v_1. It is not hard to see that this problem is solved by sorting the absolute elements of the eigenvector v_1 and keeping the support of the k entries in v_1 with the largest amplitude.

Definition 7. *Let $\mathcal{I}_k(v)$ denote the set of indices of the top k largest absolute values of a vector v.*

We can conclude that for the rank-1 case, the optimal k-sparse PC for A_1 will simply be the k-sparse vector that is co-linear to the k-sparse vector induced on this unique candidate support. This will be the only rank-1 candidate optimal support

$$\mathcal{S}_1 = \{\, \mathcal{I}_k(v_1) \,\}.$$

5.3.2 Rank-2 case

Now we describe how to compute $\mathcal{S}_2$ using the constant rank solver of [10]. This is the first nontrivial d which exhibits the details of the spannogram algorithm. We have the rank 2 matrix

$$A_2 = \sum_{i=1}^{2} \lambda_i v_i v_i^T = V_2 V_2^T,$$

where $V_2 = \left[\sqrt{\lambda_1} \cdot v_1 \;\; \sqrt{\lambda_2} \cdot v_2\right]$. We can rewrite (5.1) on A_2 as

$$\max_{x \in \mathbb{S}_k} x^T A_2 x = \max_{x \in \mathbb{S}_k} \left\| V_2^T x \right\|_2^2. \tag{5.4}$$

In the rank-1 case we could write the quadratic form maximization as a simple maximization of a dot product

$$\max_{x \in \mathbb{S}_k} x^T A_1 x = \max_{x \in \mathbb{S}_k} \left(v_1^T x \right)^2.$$

Similarly, we will prove that in the rank-2 case we can write

$$\max_{x \in \mathbb{S}_k} x^T A_2 x = \max_{x \in \mathbb{S}_k} \left(v_c^T x \right)^2,$$

for some *specific* vector v_c in the span of the eigenvectors v_1, v_2; this will be very helpful in solving the problem efficiently.

To see this, let c be a 2×1 unit length vector, i.e., $\|c\|_2 = 1$. Using the Cauchy-Schwartz inequality for the inner product of c and $V_2^T x$ we obtain $\left(c^T V_2^T x \right)^2 \leq \|V_2^T x\|_2^2$, where equality holds, if and only if, c is co-linear to $V_2^T x$. By the previous fact, we have a variational characterization of the ℓ_2-norm:

$$\|V_2^T x\|_2^2 = \max_{\|c\|_2 = 1} \left(c^T V_2^T x \right)^2. \tag{5.5}$$

We can use (B.21) to rewrite (5.4) as

$$\begin{aligned} \max_{x \in \mathbb{S}_k, \|c\|_2 = 1} \left(c^T V_2^T x \right)^2 &= \max_{x \in \mathbb{S}_k} \max_{\|c\|_2 = 1} \left(v_c^T x \right)^2 \\ &= \max_{\|c\|_2 = 1} \max_{x \in \mathbb{S}_k} \left(v_c^T x \right)^2, \end{aligned} \tag{5.6}$$

where $v_c = V_2 c$.

We would like to note two important facts here. The first is that for all unit vectors c, $v_c = V_2 c$ generates all vectors in the span of V_2 (up to scaling factors). The second fact is that if we fix c, then the maximization $\max_{x \in \mathbb{S}_k} \left(v_c^T x\right)^2$ is a rank-1 instance, similar to (5.3). Therefore, for each fixed unit vector c there will be one candidate support (denote it by $\mathcal{I}_k(V_2 c)$) to be added in $\mathcal{S}_2$.

If we could collect all possible candidate supports $\mathcal{I}_k(V_2 c)$ in

$$\mathcal{S}_2 = \bigcup_{c \in \mathbb{R}^{2\times 1}, \|c\|_2 = 1} \left\{\mathcal{I}_k(V_2 c)\right\}, \tag{5.7}$$

then we could solve exactly the sparse PCA problem on A_2: we would simply need to test all locally optimal solutions obtained from each support in $\mathcal{S}_2$ and keep the one with the maximum metric. The issue is that there are infinitely many v_c vectors to check. Naively, one could think that all possible $\binom{n}{k}$ k-supports could appear for some v_c vector. The key combinatorial fact is that if a vector v_c lives in a two dimensional subspace, there are tremendously fewer possible supports[4]:

$$|\mathcal{S}_2| \leq 4\binom{n}{2}.$$

5.3.2.1 Spherical variables and the spannogram

Here we use a transformation of our problem space into a 2-dimensional space as was done in [60]. The transformation is performed through spherical variables that enable us to visualize the 2-dimensional span of V_2. For the rank-2 case, we have a single phase variable $\phi \in \Phi = \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$ and use it to rewrite c, without loss of generality, as

$$c = \begin{bmatrix} \sin\phi \\ \cos\phi \end{bmatrix},$$

which is again unit norm and for all ϕ it scans all[5] 2×1 unit vectors. Under this characterization, we can express v_c in terms of ϕ as

$$v(\phi) = V_2 c = \sin\phi \cdot \sqrt{\lambda_1} v_1 + \cos\phi \cdot \sqrt{\lambda_2} v_2. \tag{5.8}$$

[4]This is a special case of the general d dimensional lemma of [10] (found in Appendix 5), but we prove the special case to simplify the presentation.

[5]Note that we restrict ourselves to $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$, instead of the whole $(-\pi, \pi]$ angle region. First observe that the vectors in the complement of Φ are opposite to the ones evaluated on Φ. Omitting the opposite vectors poses no issue due to the squaring in (5.4), i.e., vectors c and $-c$ map to the same solutions.

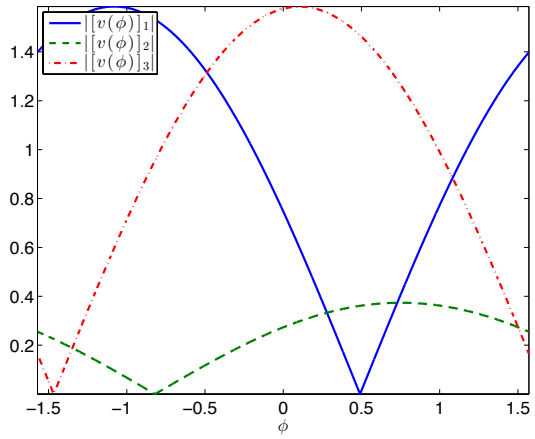


Figure 5.1: A rank-2 spannogram for a V_2 matrix with $n = 3$.

Observe that each element of $v(\phi)$ is a continuous curve in ϕ:

$$[v(\phi)]_i = \left[\sqrt{\lambda_1}v_1\right]_i \sin(\phi) + \left[\sqrt{\lambda_2}v_2\right]_2 \cos(\phi),$$

for all $i = 1, \ldots, n$. Therefore, the support set of the k largest absolute elements of $v(\phi)$ (i.e., $\mathcal{I}_k(v(\phi))$) is itself a function of ϕ.

In Fig. 6.1, we draw an example plot of 3 (absolute) curves $|[v(\phi)]_i|$, $i = 1, 2, 3$, from a randomly generated matrix V_2. We call this a *spannogram*, because at each ϕ, the values of curves correspond to the absolute values of the elements in the column span of V_2. Computing $[v(\phi)]_i$ for all i, ϕ is equivalent to computing the span of V_2. From the spannogram in Fig. 6.1, we can see that the continuity of the curves implies a local invariance property of the support sets $\mathcal{I}(v(\phi))$, around a given ϕ. As a matter of fact, a support set $\mathcal{I}_k(v(\phi))$ changes, *if and only if*, the respective sorting of two absolute elements $|[v(\phi)]_i|$ and $|[v(\phi)]_j|$ changes. Finding these interesection points $|[v(\phi)]_i| = |[v(\phi)]_j|$ is the key to find all possible support sets.

There are n curves and each pair intersects on exactly two points.[6] Therefore, there are exactly $2\binom{n}{2}$ intersection points. The intersection of two absolute

[6] As we mentioned, we assume that the curves are in "general position," i.e., no three curves intersect at the same point and this can be enforced by a small perturbation argument presented in Appendix 5.

curves are *exactly* two points ϕ that are a solution to $[v(\phi)]_i = [v(\phi)]_j$ and $[v(\phi)]_i = -[v(\phi)]_j$. These are the *only* points where local support sets might change. These $2\binom{n}{2}$ intersection points partition Φ in $2\binom{n}{2}+1$ regions within which the top k support sets remain invariant.

5.3.2.2 Building $\mathcal{S}_2$

To build $\mathcal{S}_2$, we need to *i)* determine all c intersection vectors that are defined at intersection points on the ϕ-axis and *ii)* compute all distinct locally optimal support sets $\mathcal{I}_k(v_c)$. To determine an intersection vector we need to solve all $2\binom{n}{2}$ equations $[v(\phi)]_i = \pm[v(\phi)]_j$ for all pairs $i, j \in [n]$. This yields $[v(\phi)]_i = \pm[v(\phi)]_j \Rightarrow e_i^T V c = \pm e_j^T V c$, that is

$$\left(e_i^T \pm e_j^T\right) Vc = 0 \Rightarrow c = \text{nullspace}\left(\left(e_i^T \pm e_j^T\right) V\right). \tag{5.9}$$

Since c needs to be unit norm, we simply need to normalize the solution c. We will refer to the intersection vector calculated on the ϕ of the intersection of two curves i and j as $c_{i,j}^+$ and $c_{i,j}^-$, depending on the corresponding sign in (5.9). For the intersection vectors $c_{i,j}^+$ and $c_{i,j}^-$ we compute $\mathcal{I}_k(V_2 c_{i,j}^+)$ and $\mathcal{I}_k(V_2 c_{i,j}^-)$. Observe that since the i and j curves are equal on the intersection points, there is no prevailing sorting among the two corresponding elements i and j of $V_2 c_{i,j}^+$ or $V_2 c_{i,j}^-$. Hence, for each intersection vector $c_{i,j}^+$ and $c_{i,j}^-$, we create two candidate support sets, one where element i is larger than j, and vice versa. This is done to secure that both support sets, left and right of the ϕ of the intersection, are included in $\mathcal{S}_2$. With the above methodology, we can compute all possible $\mathcal{I}_k(V_2 c)$ rank-2 optimal candidate sets and we obtain

$$|\mathcal{S}_2| \leq 4\binom{n}{2} = O(n^2).$$

The time complexity to build $\mathcal{S}_2$ is then equal to sorting $\binom{n}{2}$ vectors and solving $2\binom{n}{2}$ equations in the 2 unknowns of $c_{i,j}^+$ and $c_{i,j}^+$. That is, the total complexity is equal to $\binom{n}{2} n \log n + \binom{n}{2} 2^2 = O\left(n^3 \log n\right)$.

Remark 11. *The spannogram algorithm operates by simply solving systems of equations and sorting vectors. It is not iterative nor does it attempt to solve a convex optimization problem. Further, it computes solutions that are exactly k-sparse, where the desired sparsity can be set a-priori.*

The spannogram algorithm presented here is a subroutine that can be used to find the leading sparse PC of A_d in polynomial time. The general rank-d case is given as Algorithm 2. The details of the algorithm, the elimination step, and tune-ups for matrices with nonnegative entries can be found in Appendix 5.

5.3.3 General rank-d case

In [10], the authors show that the same ideas can be generalized for the construction of the general d case where the following holds.

Lemma 9 ([10]). *The rank-d optimal set $\mathcal{S}_d$ has $O(n^d)$ candidate optimal solutions and can be build in time $O(n^{d+1} \log n)$.*

A detailed proof of correctness is given in [10], however we provide one in Appendix Subsection B.1 for completeness.

5.4 Experimental Evaluation

We now empirically evaluate the performance of our algorithm and compare it to the full regularization path greedy approach (FullPath) of [31], the generalized power method (GPower) of [57], and the truncated power method (TPower) of [107]. We omit the DSPCA semidefinite approach of [30], since the FullPath algorithm is experimentally shown to have similar or better performance [28].

We start with a synthetic experiment: we seek to estimate the support of the first two sparse eigenvectors of a covariance matrix from sample vectors. We continue with testing our algorithm on gene expression data sets. Finally, we run experiments on a large-scale document-term data set, comprising of millions of Twitter posts.

5.4.1 Spiked Covariance Recovery

We first test our approximation algorithm on an artificial data set generated in the same manner as in [91, 107]. We consider a covariance matrix Σ, which has two sparse eigenvectors with very large eigenvalues and the rest of the eigenvectors correspond to small eigenvalues. Here, we consider $\Sigma = \sum_{i=1}^{n} \lambda_i v_i v_i^T$ with $\lambda_1 = 400, \lambda_2 = 300, \lambda_3 = 1, \ldots, \lambda_{500} = 1$. where the first two eigenvectors are sparse

and each has 10 nonzero entries and non-overlapping supports. The remaining eigenvectors are picked as $n-2$ orthogonal vectors in the nullspace of $[v_1\ v_2]$.

We have two sets of experiments, one for few samples and one for extremely few. First, we generate $m = 50$ samples of length $n = 500$ distributed as zero mean Gaussian with covariance matrix Σ and repeat the experiment 5000 times. We repeat the same experiment for $m = 5$. We compare our rank-1 and rank-2 algorithms against FullPath, GPower with ℓ_1 penalization and ℓ_0 penalization, and TPower. After estimating the first eigenvector with $\tilde{v}_1$, we deflate A to obtain A'. We use the projection deflation method [67] to obtain $A' = (I - \tilde{v}_1\tilde{v}_1^T)A(I - \tilde{v}_1\tilde{v}_1^T)$ and work on it to obtain $\tilde{v}_2$, the second estimated eigenvector of Σ.

In Table 1, we report the probability of correctly recovering the supports of v_1 and v_2: if both estimates $\tilde{v}_1$ and $\tilde{v}_2$ have matching supports with the true eigenvectors, then the recovery is considered successful.

		500×50	500×5
	k	$p_{\text{rec.}}$	$p_{\text{rec.}}$
PCA+thresh.	10	.98	0.85
GPower-ℓ_0 ($\gamma = 0.8$)	10	1	0.33
GPower-ℓ_1 ($\gamma = 0.8$)	10	1	0.33
FullPath	10	1	0.96
TPower	10	1	0.96
Rank-2 approx.	10	1	0.96

Table 5.1: Performance results on the spiked covariance model, where $p_{\text{rec.}}$ represents the recovery probability of the correct supports of the two sparse eigenvectors of Σ.

In our experiments for $m = 50$, all algorithms were comparable and performed near-optimally, apart from the rank-1 approximation (PCA+thresholding). The success of our rank-2 algorithm can be in parts suggested by the fact that the true covariance Σ is almost rank 2: it has very large decay between its 2nd and 3rd eigenvalue. The average approximation guarantee that we obtained from the generating experiments for the rank 2 case and for $m = 50$ was $x_2^T A x_2 \geq 0.7 \cdot x_* A x_*^T$, that is before running our algorithm, we know that it could on average perform at least 70% as good as the optimal solution. For $m = 5$ samples we observe that the performance of the rank-1 and GPower methods decay and FullPath, TPower, and rank-2 find the correct support with probability approximately equal to 96%.

This overall decay in performance of all schemes is due to the fact that 5 samples are not sufficient for a perfect estimate. Interesting tradeoffs of sample complexity and probability of recovery where derived in [7]. Conducting a theoretical analysis for our scheme under the spiked covariance model is left as an interesting future direction.

5.4.2 Gene Expression Data Set

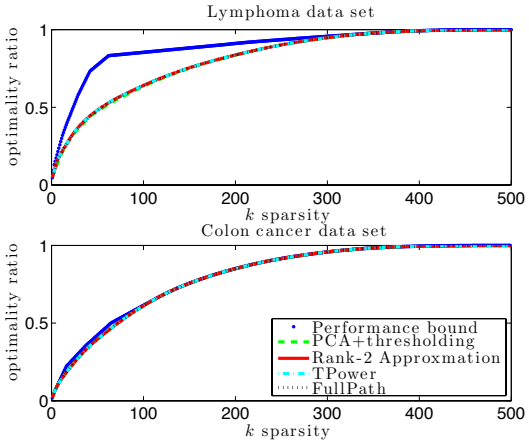


Figure 5.2: Results on gene expression data sets.

In the same manner as in the relevant sparse PCA literature, we evaluate our approximation on two gene expression data-sets used in [28, 31, 107]. We plot the ratio of the explained variance coming from the first sparse PC to the explained variance of the first eigenvector (which is equal to the first eigenvalue). We also plot the performance outer bound derived in [28]. We observe that our approximation follows the same optimality pattern as most previous methods, for many values of sparsity k. In these experiments we did not test the GPower method since the output sparsity cannot be explicitly predetermined. However, previous literature indicates that GPower is also near-optimal in this scenario.

1st sparse PC				
Rank-1	**TPower**	**Rank-**2	**Rank-**3	**FullPath**
skype	~~**eurovision**~~	skype	skype	eurovision
microsoft	skype	microsoft	microsoft	finalG
billion	microsoft	billion	acquisitionG	greeceG
acquisitionG	billion	acquisitionG	billion	greece
~~**eurovision**~~	acquisitionG	acquiredG	acquiredG	lucasG
acquiredG	buying	acquiresG	acquiresG	semifinalG
acquiresG	acquiredG	buying	buying	final
buying	acquiresG	dollarsG	dollarsG	contest
google	dollarsG	acquisition	acquisition	stereo
dollarsG	acquisition	google	google	watching
performance = $\frac{\text{explained variance}}{\text{maximum explained variance}} = \frac{x_1^T A x_1}{\lambda_1}$				
0.9863	0.9861	**0.9870**	**0.9870**	0.9283

2nd sparse PC				
Rank-1	**TPower**	**Rank-**2	**Rank-**3	**FullPath**
greece	greece	eurovision	eurovision	skype
greeceG	greeceG	greece	greece	microsoft
love	love	greeceG	lucasG	billion
lucasG	loukas	finalG	finalG	acquisitionG
final	finalsG	lucasG	final	acquiresG
greek	~~**athens**~~	final	stereo	acquiredG
~~**athens**~~	final	stereo	semifinalG	buying
finalG	stereo	semifinalG	contest	dollarsG
stereo	country	contest	greeceG	official
country	~~**sailing**~~	songG	watching	google
performance = $\frac{\text{explained variance}}{\text{maximum explained variance}} = \frac{\sum_{i=1}^{2} x_i^T A x_i}{\sum_{i=1}^{2} \lambda_i}$				
0.8851	0.8850	0.9850	**0.9852**	**0.9852**

3rd sparse PC				
Rank-1	**TPower**	**Rank-**2	**Rank-**3	**FullPath**
~~**downtown**~~G	~~**twitter**~~	love	love	love
censusG	censusG	received	received	received
~~**athensG**~~	homeG	greek	twitter	damon
homeG	~~**google**~~	~~**know**~~	~~**know**~~	greek
~~**twitter**~~	yearG	damon	greek	hate
yearG	greek	amazing	damon	~~**know**~~
~~**murderG**~~	mayG	hate	hate	amazing
~~**songG**~~	~~**facebook**~~	twitter	amazing	sweet
mayG	startsG	great	great	great
yearsG	populationG	sweet	sweet	~~**songs**~~
performance = $\frac{\text{explained variance}}{\text{maximum explained variance}} = \frac{\sum_{i=1}^{3} x_i^T A x_i}{\sum_{i=1}^{3} \lambda_i}$				
0.7875	0.7877	0.8993	**0.8994**	**0.8994**

Table 5.2: The first 3 (out of 5) sparse PCs for a data-set consisting of 65k Tweets and 64k unique words. Words that appear with a G are translated from Greek.

4th sparse PC				
Rank-1	**TPower**	**Rank-2**	**Rank-3**	**FullPath**
thanouG	downtownG	downtownG	downtownG	~~**twitter**~~
kenterisG	athensG	athensG	athensG	~~**facebook**~~
guiltyG	yearG	murderG	murderG	~~**welcome**~~
kenteris	year'sG	yearsG	yearsG	~~**account**~~
tzekosG	murderG	brutalG	brutalG	~~**goodG**~~
monthsG	cameraG	stabbedG	stabbedG	~~**followers**~~
tzekos	crimeG	bad_eventsG	bad_eventsG	censusG
~~**facebook**~~	crime	yearG	cameraG	populationG
imprisonmentG	stabbedG	turmoilG	yearG	homeG
penaltiesG	brutalG	cameraG	crimeG	startsG
performance = $\frac{\text{explained variance}}{\text{maximum explained variance}} = \frac{\sum_{i=1}^{4} x_i^T A x_i}{\sum_{i=1}^{4} \lambda_i}$				
0.7174	0.7520	0.8419	**0.8420**	0.8412
5th sparse PC				
Rank-1	**TPower**	**Rank-2**	**Rank-3**	**FullPath**
bravoG	songG	censusG	censusG	~~**yearG**~~
loukaG	bravoG	homeG	homeG	~~**this_yearG**~~
~~**athensG**~~	~~**endG**~~	populationG	populationG	loveG
~~**endG**~~	loukaG	may'sG	may'sG	~~**birthdayG**~~
~~**woman**~~G	likedG	beginsG	beginsG	~~**i_wishG**~~
successG	niceG	generalG	general	songG
niceG	greekG	nightG	begunG	titleG
~~**youtube**~~	titleG	~~**noneG**~~	comesG	~~**memoriesG**~~
~~**was_goingG**~~	~~**trials**~~G	yearG	census_employeeG	~~**trialsG**~~
~~**murderedG**~~	~~**memoriesG**~~	countryG	yearG	likedG
performance = $\frac{\text{explained variance}}{\text{maximum explained variance}} = \frac{\sum_{i=1}^{5} x_i^T A x_i}{\sum_{i=1}^{5} \lambda_i}$				
0.6933	0.7464	0.8343	**0.8345**	0.8241

Table 5.3: The remaining 2 (out of 5) sparse PCs for a data-set consisting of 65k Tweets and 64k unique words. Words that appear with a G are translated from Greek.

5.4.3 Large-scale Twitter data-set

We proceed our experimental evaluation of our algorithm by testing it on a large-scale data set. Our data-set comprises of millions of tweets coming from Greek Twitter users. Each tweet corresponds to a list of words and has a character limit of 140 per tweet. Although each tweet was associated with metadata, such us hyperlinks, user id, hash tags etc., we strip these features out and just use the word list. We use a simple Python script to normalize each Tweet. Words that are not contextual (`me, to, what,` etc) are discarded in an ad-hoc way. We also discard all words that are less than three characters, or words that appear once in the corpus. We represent each tweet as a long vector consisting of n words, with a 1 whenever a word appears, and 0 if it does not appear. Further details about our data set can be found in Appendix 5.

Document-term data sets have been observed to follow power-laws on their eigenvalues. Empirical results have been reported that indicate power-law like decays for eigenvalues where no cutoff is observed [33] and some derived power-law generative models for $0/1$ matrices [25, 70]. In our experiments, we also observe power-law decays on the spectrum of the twitter matrices. Further experimental observations of power laws can be found in Appendix 5. These underlying decay laws on the spectrum were sufficient to give good approximation guarantees; for many of our data sets $1 - \epsilon$ was between 0.5 to 0.7, even for $d = 2, 3$. Further, our algorithm empirically performed better than these guarantees.

In the following tests, we compare against TPower and FullPath. TPower is run for $10k$ iterations, and is initialized with a vector having 1s on the k words of highest variance. For FullPath we restrict the covariance to its first 5k words of highest variance, since for larger numbers the algorithm became slow to test on a personal desktop computer. In our experiments, we use a simpler deflation method, than the more sophisticated ones used before. Once k words appear in the first k-sparse PC, we strip them from the data set, recompute the new convariance, and then run all algorithms. A benefit of this deflation is that it forces all sparse PCs to be orthogonal to each other which helps for a more fair comparison with respect to explained variance. Moreover, this deflation preserves the sparsity of the matrix A after each deflation step; sparsity on A facilitates faster execution times for all methods tested. The performance metric here is again the explained variance over its maximum possible value: if we compute L PCs, $x_1, \ldots, x_L$, we measure their

performance as $\frac{\sum_{i=1}^{L} x_i^T A x_i}{\sum_{i=1}^{L} \lambda_i}$. We see that in many experiments, we come very close to the optimal value of 1.

In Table 5.4, we show our results for all tweets that contain the word `Japan`, for a 5-day (May 1-5, 2011) and then a month-length time window (May, 2011). In all these tests, our rank-3 approximation consistently captured more variance than all other compared methods.

	*japan	1-5 May 2011	May 2011
$m \times n$	12k $\times$ 15k	267k $\times$ 148k	1.9mil $\times$ 222k
k	$k = 10$	$k = 4$	$k = 5$
#PCs	5	7	3
Rank-1	0.600	0.815	0.885
TPower	0.595	0.869	0.915
Rank-2	**0.940**	0.934	0.885
Rank-3	**0.940**	**0.936**	**0.954**
FullPath	0.935	0.886	0.953

Table 5.4: Performance comparison on the Twitter data-set

In Table 5.3, we show a day-length experiment (May 10th, 2011), where we had 65k Tweets and 64k unique words. For this data-set we report the first 5 sparse PCs generated by all methods tested. The average computation times for this time-window where less than 1 second for the rank-1 approximation, less than 5 seconds for rank-2, and less than 2 minutes for the rank-3 approximation on a Macbook Pro 5.1 running MATLAB 7. The main reason for these tractable running times is the use of our elimination scheme which left only around $40 - 80$ rows of the initial matrix of 64k rows. In terms of running speed, we empirically observed that our algorithm is slower than Tpower but faster than FullPath for the values of d tested. In Table 5.3, words with strike-through are what we consider non-matching to the "main topic" of that PC. Words marked with G are translated from Greek. From the PCs we see that the main topics are about Skype's acquisition by Microsoft, the European Music Contest "Eurovision," a crime that occurred in the downtown of Athens, and the Greek census that was carried for the year 2011. An interesting observation is that a general "excitement" sparse principal component appeared in most of our queries on the Twitter data set. It involves words like `like, love, liked, received, great`, etc, and was generated by all algorithms.

5.5 Conclusions

We conclude that our algorithm can efficiently provide interpretable sparse PCs while matching or outperforming the accuracy of previous methods. A parallel implementation in the MapReduce framework and larger data studies are very interesting future directions.

Chapter 6

Finding Dense Subgraphs via Low-Rank Bilinear Optimization

In this chapter, we develop a novel algorithm for DkS that searches a low-dimensional space for provably good solutions. We obtain provable performance bounds that depend on the graph spectrum. We establish data-dependent approximation bounds, related to the spectrum of the adjacency matrix. This is important, as existing results have only provided worst-case guarantees – far too conservative to provide guidance on real-world problems. Our algorithm exploits a combinatorial structure called the *Spannogram* originally introduced in [10], related to a low-rank approximation of the adjacency matrix.

Our algorithm is highly suitable for parallel implementation: we implement it in MapReduce, run it on a large computer cluster, and test it on artificial and real data sets. We show that it is possible to find dense subgraphs in massive graphs that have billions of edges. We empirically show that our algorithm can find subgraphs of significantly higher density compared to the previous state of the art. [1]

6.1 Introduction

Given a graph $\mathcal{G}$ on n vertices with m edges and a parameter k, we are interested in finding an induced subgraph on k vertices with the largest average degree, also known as the *maximum density*. This is the *Densest k-Subgraph* (DkS)

[1] Contributions Statement: Most parts of this chapter are accepted for presentation in the International Conference on Machine Learning, in 2014, under the title, Finding Dense Subgraphs via Low-Rank Bilinear Optimization, with authors, D.S. Papailiopoulos, I. Mitlagkas, A. G. Dimakis, C. Caramanis. Prof. Alex Dimakis and Prof. Constantine Caramanis supervised the project and all coauthors had equal contribution to this work.

– a fundamental problem in combinatorial optimization with applications in numerous fields including social sciences, communication networks, and biology (see *e.g.* [11,37,42,49,71,87]).

DkS is a notoriously hard problem. It is NP-hard by reduction to MAX-CLIQUE. Moreover, Khot showed in [62] that, under widely believed complexity-theoretic assumptions, DkS cannot be approximated within an arbitrary constant factor.[2] The best known approximation ratio was $n^{1/3+\epsilon}$ (for some small ϵ) due to [40]. Recently, [15] introduced an algorithm with approximation ratio $n^{1/4+\epsilon}$, that runs in time $n^{O(1/\epsilon)}$. Such results, where the approximation factor scales as a polynomial in the number of vertices, are too pessimistic for real-world applications. This resistance to better approximations, despite the long history of the problem, suggests that DkS is probably very hard in the worst case.

Our Contributions. In this work we move beyond the worst case framework. We present a novel DkS algorithm that has two key features: *i)* it comes with approximation guarantees that are surprisingly tight on real-world graphs and *ii*) it is fully parallelizable and can scale up to graphs with billions of edges.

Our algorithm combines spectral and combinatorial techniques; it relies on examining candidate subgraphs obtained from vectors lying in a low-dimensional subspace of the adjacency matrix of the graph. This is accomplished through a framework called the *Spannogram*, introduced in [60] and later used in [10], which we define below, and is very similar to the one of the previous chapter.

Our approximation guarantees are *graph-dependent*: they are related to the spectrum of the adjacency matrix of the graph. Let opt denote the average degree (i.e., the density) of the densest k-subgraph, where $0 \leq \mathsf{opt} \leq k-1$. Our algorithm takes as input the graph, the subgraph size k, and an accuracy parameter $d \in \{1, \ldots, n\}$. The output is a subgraph on k vertices with density opt_d, for which we obtain the following approximation result:

Theorem 1. *For any unweighted graph, our algorithm outputs in time*

$$O\left(\frac{n^{d+2} \cdot \log n}{\delta}\right)$$

[2] approximation ratio ρ means that there exists an algorithm that produces in polynomial time a number A, such that $1 \leq \frac{\mathsf{opt}}{\mathsf{A}} \leq \rho$, where opt is the optimal density.

a k-subgraph that has density

$$\mathsf{opt}_d \geq 0.5 \cdot (1-\delta) \cdot \mathsf{opt} - 2 \cdot |\lambda_{d+1}|,$$

with probability $1 - \frac{1}{n}$, where λ_i is the ith largest, in magnitude, eigenvalue of the adjacency matrix of the graph. If the graph is bipartite, or if the largest d eigenvalues of the graph are positive, then our algorithm runs in time $O\left(n^{d+1} + \mathsf{T}_d\right)$, and outputs a k-subgraph with density

$$\mathsf{opt}_d \geq \mathsf{opt} - 2 \cdot |\lambda_{d+1}|,$$

where T_d is the time to compute the d leading eigenvectors of the adjacency matrix of the graph.

Our bounds come close to $2+\epsilon$ and $1+\epsilon$ factor approximations, when λ_{d+1} is significantly smaller than the density of the densest k-subgraph. In the following theorem, we give such an example. However, we would like to note that in the worst case our bounds might not yield something meaningful.

Theorem 2. *If the densest-k-subgraph contains a constant fraction of all the edges, and $k = \Theta(\sqrt{E})$, then we can approximate DkS within a factor of $2+\epsilon$, in time $n^{O(1/\epsilon^2)}$. If additionally the graph is bipartite, we can approximate DkS within a factor of $1+\epsilon$.*

The above result is similar to the $1+\epsilon$ approximation ratio of [8] for dense graphs, where the densest-k-subgraph contains a constant fraction of the $\Omega(n^2)$ edges, where $k = \Omega(n)$. The innovation here is that our ratio also applies to *sparse graphs* with sublinear number of edges.

Computable upper bounds. In addition to these theoretical guarantees, our analysis allows us to obtain a graph-dependent upper bound for the optimal subgraph density. This is shown in Fig. 3 in our experimental section, where for many graphs our algorithm is provably within 70% from the upper bound of opt. These are far stronger guarantees than the best available *a priori* bounds. This illustrates the potential power of graph-dependent guarantees that, however, require the execution of an algorithm.

Nearly-linear time approximation. Our algorithm has a worst-case running time of $O\left(\frac{n^{d+2} \cdot \log n}{\delta}\right)$. Under some mild spectral assumptions, a randomized version of our algorithm runs in nearly-linear time.

Theorem 3. *Let the d largest eigenvalues of the graph be positive, and let the d-th,$(d+1)$-st largest have constant ratio:*

$$\left|\frac{\lambda_d}{\lambda_{d+1}}\right| \geq C.$$

Then, we can modify our algorithm to output, with probability $1-\delta$, a k-subgraph with density

$$(1-\epsilon)^2 \cdot \mathsf{opt}_d,$$

in time

$$O\left(m \cdot \log n + \frac{n}{\epsilon^d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right),$$

where m is the number of edges.

We found that the above spectral condition holds for all $d \leq 5$, in many real-world graphs that we tested.

Scalability. We develop two key scalability features that allow us to scale up efficiently on massive graphs.

Vertex sparsification: We introduce a pre-processing step that eliminates vertices that are unlikely to be part of the densest k-subgraph. The elimination is based on the vertices' *weighted leverage scores* [16, 68] and admits a provable bound on the introduced error. We empirically found that even with a negligible additional error, the elimination dramatically reduced problem sizes in all tested datasets.

MapReduce implementation: We show that our algorithm is *fully-parallelizable* and tailor it for the MapReduce framework. We use our MapReduce implementation to run experiments on Elastic MapReduce (EMR) on Amazon. In our large-scale experiments, we were able to scale out to thousands of mappers and reducers in parallel over 800 cores, and find large dense subgraphs in graphs with billions of edges.

6.1.1 Related work

DkS algorithms: One of the few positive results for DkS is a $1+\epsilon$ approximation for dense graphs where $m = \Omega(n^2)$, and in the linear subgraph setting $k = \Omega(n)$ [8]. For some values of $m = o(n^2)$ a $2+\epsilon$ approximation was established by [97]. Moreover, for any $k = \Omega(n)$ a constant factor approximation is possible

via a greedy approach by [9], or via semidefinite relaxations by [94] and [39]. Recently, [5] established new approximation results for graphs with small "ϵ-rank," using an approximate solver for low-rank perturbed versions of the adjacency matrix.

There is a vast literature on algorithms for detecting communities and well-connected subgraphs: greedy schemes [84], optimization approaches [6,32,53], and the truncated power method [107]. We compare with various of these algorithms in our evaluation section.

The Spannogram framework: We present an exact solver for *bilinear* optimization problems on matrices of constant rank, under $\{0,1\}$ *and* sparsity constraints on the variables. Our theory is a generalization of the Spannogram framework, originally introduced in the foundational work of [60] and further developed in [10,79], that obtains exact solvers for low-rank *quadratic optimization* problems with combinatorial constraints, such as sparse PCA.

MapReduce algorithms for graphs: The design of MapReduce algorithms for massive graphs is an active research area as Hadoop becomes one of the standards for storing large data sets. The related work by Bahmani *et al.* [11] designs a novel MapReduce algorithm for the *densest subgraph* problem. This densest subgraph problem requires finding a subgraph of highest *normalized density* without enforcing a specific subgraph size k. Surprisingly, without a subgraph size restriction, the densest subgraph becomes polynomially solvable and therefore fundamentally different from what we consider in this paper.

6.2 Proposed Algorithm

The *density* of a subgraph indexed by a vertex set $\mathcal{S} \subseteq \{1,\ldots,n\}$ is equal to the average degree of the vertices within $\mathcal{S}$:

$$\mathsf{den}(\mathcal{S}) = \frac{\mathbf{1}_{\mathcal{S}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}}}{|\mathcal{S}|}$$

where $\mathbf{A}$ is the adjacency matrix ($A_{i,j} = 1$ if (i,j) is an edge, else $A_{i,j} = 0$) and the indicator vector $\mathbf{1}_{\mathcal{S}}$ has 1s in the entries indexed by $\mathcal{S}$ and 0 otherwise. Observe that $\mathbf{1}_{\mathcal{S}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}} = \sum_{i,j \in \mathcal{S}} A_{i,j}$ is twice the number of edges in the subgraph with vertices in $\mathcal{S}$.

For a fixed subgraph size $|\mathcal{S}| = k$, we can express DkS as a quadratic optimization:

$$\mathsf{DkS}: \quad \mathsf{opt} = (1/k) \cdot \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}}$$

where $|\mathcal{S}| = k$ denotes that the optimization variable is a k-vertex subset of $\{1, \ldots, n\}$.

The bilinear relaxation of DkS. We approximate DkS via approximating its bipartite version. This problem can be expressed as a bilinear maximization:

$$\mathsf{DBkS}: \quad \mathsf{opt}^{\mathsf{B}} = (1/k) \cdot \max_{|\mathcal{X}|=k} \max_{|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}}.$$

As we see in the following lemma, the two problems are fundamentally related: a good solution for the bipartite version of the problem maps to a "half as good" solution for DkS. The proof is given in Appendix B.

Lemma 10. *A ρ-approximation algorithm for DBkS implies a 2ρ-approximation algorithm for DkS.*

6.2.1 DkS through low rank approximations

At the core of our approximation lies a constant rank solver: we show that DBkS can be solved in polynomial time on constant rank matrices. We solve constant rank instances of DBkS instead of DkS due to an important implication: DkS is NP-hard even for rank-1 matrices with 1 negative eigenvalue, as we show in Appendix B.

The exact steps of our algorithm are given in the pseudo-code tables referred to as Algorithms 1-3.[3] The output of our algorithm is a k-subgraph $\mathcal{Z}_d$ that has density opt_d that comes with provable guarantees. We present our theoretical guarantees in the next subsection.

Our main algorithmic innovation, the constant rank solver for DBkS (Algorithms 2-3), is called many times: in lines 5, 8, and 15 of our general DkS approximation, shown as Algorithm 1. We describe its steps subsequently.

[3] In the pseudocode of Algorithm 2, $\text{top}_k(\mathbf{v})$, denotes the indices of the k largest signed elements of $\mathbf{v}$.

Algorithm 2 low-rank approximations for DkS

$[\mathbf{V}_d, \Lambda_d] = \texttt{EVD}(\mathbf{A}, d)$
if $\mathcal{G}$ is bipartite **then**
  $\mathbf{B}$ = bi-adjacency of $\mathcal{G}$
  $[\mathbf{V}_d, \Sigma_d, \mathbf{U}_d] = \texttt{SVD}(\mathbf{B}, d)$
  $\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg\max_{|\mathcal{X}|+|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \Sigma_d \mathbf{U}_d^T \mathbf{1}_{\mathcal{Y}}$.
  $\mathcal{Z}_d = \mathcal{X}_d \cup \mathcal{Y}_d$
else if The first d eigenvalues of $\mathbf{A}$ are positive **then**
  $\{\mathcal{X}_d, \mathcal{X}_d\} = \arg\max_{|\mathcal{X}|=|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \Lambda_d \mathbf{V}_d^T \mathbf{1}_{\mathcal{Y}}$.
  $\mathcal{Z}_d = \mathcal{X}_d$
else
  for $i = 1 : \frac{\log n}{\delta}$ **do**
    draw n fair coins and assign them to vertices
    $\mathcal{L}$ = vertices with heads; $\mathcal{R} = \{1, \ldots, n\} - \mathcal{L}$
    $\mathbf{B}_d^i = [\mathbf{V}_d \Lambda_d \mathbf{V}_d^T]_{\mathcal{L},\mathcal{R}}$
    $\{\mathcal{X}^i, \mathcal{Y}^i\} = \arg\max_{|\mathcal{X}|+|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{B}_d^i \mathbf{1}_{\mathcal{Y}}$.
  end for
  $\{\mathcal{X}^i, \mathcal{Y}^i\} = \arg\max_{1 \le i \le n} \mathbf{1}_{\mathcal{X}^i}^T \mathbf{B}_d^i \mathbf{1}_{\mathcal{Y}^i}$
  $\mathcal{Z}_d = \mathcal{X}_d \cup \mathcal{Y}_d$
end if
Output: $\mathcal{Z}_d$

Constant rank solver for DBkS**.** In the following we present an exact solver for DBkS on constant rank approximations of $\mathbf{A}$. Our DkS algorithm makes a number of calls to the DBkS low-rank solver on slightly different (some times rectangular) matrices. The details of the general low-rank solver are in Appendix B.

Step 1: *Obtain* $\mathbf{A}_d = \sum_{i=1}^{d} \lambda_i \mathbf{v}_i \mathbf{v}_i^T$*, a rank-*$d$ *approximation of* $\mathbf{A}$. Here, λ_i is the i-th largest in magnitude eigenvalue and $\mathbf{v}_i$ the corresponding eigenvector.

Step 2: *Use* $\mathbf{A}_d$ *to obtain* $O(n^d)$ *candidate subgraphs.* For any matrix $\mathbf{A}$ we can solve DBkS by exhaustively checking all $\binom{n}{k}^2$ pairs $(\mathcal{X}, \mathcal{Y})$ of k-subsets of vertices. Surprisingly, if we want to find the $\mathcal{X}, \mathcal{Y}$ pairs that maximize $\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}$, i.e., the bilinear problem on the rank-d matrix $\mathbf{A}_d$, then we show that only $O(n^d)$ candidate pairs need to be examined. The proof of this is done using a modification to the analysis of [10], and essentially use the same algorithm as in [10] to recover the candidate pairs.

Step 3: Check all k-set pairs $\{\mathcal{X}, \mathcal{Y}\}$ obtained by Step 2, and output the one with the largest density on the low-rank weighted adjacency $\mathbf{A}_d$.

In the next section, we derive the constant rank-solver using two key facts. First, for each fixed vertex set $\mathcal{Y}$, we show that it is easy to find the optimal set

$\mathcal{X}$ that maximizes $\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}$ for that $\mathcal{Y}$. Since this turns out to be easy, then the challenge is to find the number of different vertex sets $\mathcal{Y}$ that we need to check. Do we need to exhaustively check all $\binom{n}{k}$ k-sets $\mathcal{Y}$? We show that this question is equivalent to searching the span of the first d eigenvectors of $\mathbf{A}$, and collecting in a set $\mathcal{S}_d$ the top-k coordinates of all vectors in that d-dimensional space. By modifying the Spannogram theory of [10,60], we show how this set has size $O(n^d)$ and can be constructed in time $O(n^{d+1})$. This will imply that DBkS can be solved in time $O(n^{d+1})$ on $\mathbf{A}_d$.

Computational Complexity. The worst-case time complexity of the constant-rank DBkS solver on $\mathbf{A}_d$ is $O(\mathsf{T}_d + n^{d+1})$, where T_d is the time to compute the first d eigenvectors of $\mathbf{A}$. Under conditions satisfied by many real world graphs, we show that we can modify our algorithm and obtain a randomized one that succeeds with probability δ and is ϵ far from the optimal rank-d solver, while its complexity reduces to *nearly linear* in the number of edges m of the graph $\mathcal{G}$: $O\left(m \cdot \log n + \frac{n}{\epsilon^d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$.

Algorithm 3 `lowrankDBkS`$(k,\ d,\ \mathbf{A})$

$[\mathbf{V}_d, \Lambda_d] = \texttt{EVD}(\mathbf{A}, d)$
$\mathcal{S}_d = \texttt{Spannogram}(k, \mathbf{V}_d)$
$\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg\max_{|\mathcal{X}|=k} \max_{\mathcal{Y} \in \mathcal{S}_d} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \Lambda_d \mathbf{V}_d^T \mathbf{1}_{\mathcal{Y}}$
Output: $\{\mathcal{X}_d, \mathcal{Y}_d\}$

`Spannogram`$(k,\ \mathbf{V}_d)$
$\mathcal{S}_d = \{\text{top}_k(\mathbf{v}) : \mathbf{v} \in \text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_d)\}$
Output: $\mathcal{S}_d$.

6.2.2 Approximation Guarantees

We approximate DBkS by finding a solution to the constant rank problem

$$\max_{|\mathcal{X}|=k} \max_{|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}.$$

We output a pair of vertex sets, $\mathcal{X}_d, \mathcal{Y}_d$, which we refer to as the *rank-d optimal solution*, that has density

$$\mathsf{opt}_d^{\mathsf{B}} = (1/k) \cdot \mathbf{1}_{\mathcal{X}_d}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_d}.$$

Our approximation guarantees measure how far $\mathsf{opt}_d^{\mathsf{B}}$ is from $\mathsf{opt}^{\mathsf{B}}$, the optimal density for DBkS. Our bounds capture a simple core idea: the loss in our approximation comes due to solving the problem on $\mathbf{A}_d$ instead of solving it on the full

rank matrix $\mathbf{A}$. This loss is quantified in the next lemma. The detailed proofs of the following results are in Appendix B.

Lemma 11. *For any matrix* $\mathbf{A}$*:* $\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot |\lambda_{d+1}|$*, where* λ_i *is the* i*th largest eigenvalue of* $\mathbf{A}$*.*

Using an appropriate pre-processing step and then running Algorithm 2 as a subroutine on a sub-sampled and low-rank version of $\mathbf{A}$, we output a k-subgraph $\mathcal{Z}_d$ that has density opt_d. By essentially combining Lemmata 1 and 2 we obtain the following bounds.

Theorem 6. *Algorithm 1 outputs in time* $O\left(\frac{n^{d+2} \cdot \log n}{\delta}\right)$ *a* k*-subgraph that has density*

$$\mathsf{opt}_d = \mathsf{den}(\mathcal{Z}_d) \geq 0.5 \cdot (1-\delta) \cdot \mathsf{opt} - 2 \cdot |\lambda_{d+1}|,$$

with probability $1 - \frac{1}{n}$*, where* λ_i *is the* i*th largest, in magnitude, eigenvalue of the adjacency matrix of the graph. If the graph is bipartite, or if the largest* d *eigenvalues of the graph are positive, then our algorithm runs in time* $O\left(n^{d+1} + \mathsf{T}_d\right)$*, and outputs a* k*-subgraph with density* $\mathsf{opt}_d \geq \mathsf{opt} - 2 \cdot |\lambda_{d+1}|$*, where* T_d *is the time to compute the* d *leading eigenvectors of the adjacency matrix of the graph.*

Using bounds on eigenvalues of graphs, Theorem 6 translates to the following approximation guarantees.

Theorem 7. *If the densest-*k*-subgraph contains a constant fraction of all the edges, and* $k = \Theta(\sqrt{E})$*, then we can approximate DkS within a factor of* $2 + \epsilon$*, in time* $n^{O(1/\epsilon^2)}$*. If additionally the graph is bipartite, then we can approximate DkS within a factor of* $1 + \epsilon$*.*

Remark 12. *The above results are similar to the* $1 + \epsilon$ *ratio of [8], which holds for graphs where the densest-*k*-subgraph contains* $\Omega(n^2)$ *edges.*

Graph dependent bounds. For any given graph, after running our constant rank solver on $\mathbf{A}_d$, we can compute an upper bound to the optimal density opt via bounds on $\mathsf{opt}^{\mathsf{B}}$, since it is easy to see that $\mathsf{opt}^{\mathsf{B}} \geq \mathsf{opt}$. Our graph-dependent bound is the minimum of three upper bounds on the unknown optimal density:

Lemma 12. *The optimal density of DkS can be bounded as*

$$\mathsf{opt} \leq \min\left\{ (1/k) \cdot \mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d} + |\lambda_{d+1}|,\ k-1,\ \lambda_1 \right\}.$$

In our experimental section, we plot the above upper bounds, and show that for most tested graphs our algorithm performs *provably* within 70% from the upper bound on the optimal density. These are far stronger guarantees than the best available *a priori* bounds.

6.3 The Spannogram Framework

In this section, we describe how the constant rank solver operates by examining candidate vectors in a low-dimensional span of $\mathbf{A}$.

Here, we work on a rank-d matrix $\mathbf{A}_d = \mathbf{v}_1\mathbf{u}_1^T + \ldots + \mathbf{v}_d\mathbf{u}_d^T$ where $\mathbf{u}_i = \lambda_i\mathbf{v}_i$, and we wish to solve:

$$\max_{|\mathcal{X}|=|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \left(\mathbf{v}_1\mathbf{u}_1^T + \ldots + \mathbf{v}_d\mathbf{u}_d^T\right) \mathbf{1}_{\mathcal{Y}}. \tag{6.1}$$

Observe that we can rewrite (6.1) in the following way

$$\begin{aligned} &\max_{|\mathcal{X}|=|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \left[\mathbf{v}_1 \cdot \underbrace{(\mathbf{u}_1^T\mathbf{1}_{\mathcal{Y}})}_{c_1} + \ldots + \mathbf{v}_d \cdot \underbrace{(\mathbf{u}_d^T\mathbf{1}_{\mathcal{Y}})}_{c_d}\right] \\ &= \max_{|\mathcal{Y}|=k} \left(\max_{|\mathcal{X}|=k} \mathbf{1}_{\mathcal{X}}^T\mathbf{v}_{\mathcal{Y}}\right), \end{aligned} \tag{6.2}$$

where $\mathbf{v}_{\mathcal{Y}} = \mathbf{v}_1 \cdot c_1 + \ldots + \mathbf{v}_d \cdot c_d$ is an n-dimensional vector generated by the d-dimensional subspace spanned by $\mathbf{v}_1, \ldots, \mathbf{v}_d$.

We will now make a key observation: for every fixed vector $\mathbf{v}_{\mathcal{Y}}$ in (6.2), the index set $\mathcal{X}$ that maximizes $\mathbf{1}_{\mathcal{X}}^T\mathbf{v}_{\mathcal{Y}}$ can be easily computed. It is not hard to see that for any fixed vector $\mathbf{v}_{\mathcal{Y}}$, the k-subset $\mathcal{X}$ that maximizes $\mathbf{1}_{\mathcal{X}}^T\mathbf{v}_{\mathcal{Y}} = \sum_{i\in\mathcal{X}}[\mathbf{v}_{\mathcal{Y}}]_i$ corresponds the set of k largest signed coordinates of $\mathbf{v}_{\mathcal{Y}}$. That is, the locally optimal k-set is $\text{top}_k(\mathbf{v}_{\mathcal{Y}})$.

We now wish to find all possible locally optimal sets $\mathcal{X}$. If we could possibly check all vectors $\mathbf{v}_{\mathcal{Y}}$, then we could find all locally optimal index sets $\text{top}_k(\mathbf{v}_{\mathcal{Y}})$.

Let us denote as $\mathcal{S}_d$ the set of all k-subsets $\mathcal{X}$ that are the optimal solutions of the inner maximization of (6.2) for *any* vector $\mathbf{v}$ in the span of $\mathbf{v}_1, \ldots, \mathbf{v}_d$

$$\mathcal{S}_d = \{\text{top}_k([\mathbf{v}_1 \cdot c_1 + \ldots + \mathbf{v}_d \cdot c_d]) : c_1, \ldots, c_d \in \mathbb{R}\}.$$

Clearly, this set contains all possible locally optimal $\mathcal{X}$ sets of the form $\text{top}_k(\mathbf{v}_\mathcal{Y})$. Therefore, we can rewrite DBkS on $\mathbf{A}_d$ as

$$\max_{|\mathcal{Y}|=k} \max_{\mathcal{X}\in\mathcal{S}_d} \mathbf{1}_\mathcal{X}^T \mathbf{A}_d \mathbf{1}_\mathcal{Y}. \tag{6.3}$$

The above problem can now be solved in the following way: for every set $\mathcal{X} \in \mathcal{S}_d$ find the locally optimal set $\mathcal{Y}$ that maximizes $\mathbf{1}_\mathcal{X}^T \mathbf{A}_d \mathbf{1}_\mathcal{Y}$, that is, this will be $\text{top}_k(\mathbf{A}_d \mathbf{1}_\mathcal{X})$. Then, we simply need to test all such $\mathcal{X}, \mathcal{Y}$ pairs on $\mathbf{A}_d$ and keep the optimizer.

Due to the above, the problem of solving DBkS on $\mathbf{A}_d$ is equivalent to constructing the set of k-supports $\mathcal{S}_d$, and then finding the optimal solution in that set. How large can $\mathcal{S}_d$ be and can we construct it in polynomial time? Initially one could expect that the set $\mathcal{S}_d$ could have size as big as $\binom{n}{k}$. Instead, we show that the set $\mathcal{S}_d$ will be tremendously smaller, as in [60] and [10].

Lemma 13. *The set $\mathcal{S}_d$ has size at most $O(n^d)$ and can be built in time $O(n^{d+1})$ using Algorithm 2.*

The above lemma is proved using the same technique as in [10].

6.3.1 Constructing the set $\mathcal{S}_d$

We build up to the general rank-d algorithm by explaining special cases that are easier to understand.

Rank-1 case. We start with the $d = 1$ case, where we have $\mathcal{S}_1 = \{\text{top}_k(c_1 \cdot \mathbf{v}_1) : c_1 \in \mathbb{R}\}$. It is not hard to see that there are only two supports to include in $\mathcal{S}_1$: $\text{top}_k(\mathbf{v}_1)$ and $\text{top}_k(-\mathbf{v}_1)$. These two sets can be constructed in time in time $O(n)$, via a partial sorting and selection algorithm [26]. Hence, $\mathcal{S}_1$ has size 2 and can be constructed in time $O(n)$.

Rank-2 case. This is the first non-trivial d which exhibits the details of the Spannogram algorithm. The following analysis is the same as [10].

Let an auxiliary angle $\phi \in \Phi = [0,\ \pi)$ and let

$$\mathbf{c} = \left[\begin{smallmatrix} c_1 \\ c_2 \end{smallmatrix}\right] = \left[\begin{smallmatrix} \sin\phi \\ \cos\phi \end{smallmatrix}\right].^4$$

[4]Observe that when we scan ϕ, the vectors $\mathbf{c}, -\mathbf{c}$ express all possible unit norm vectors on the circle.

Then, we re-express $c_1 \cdot \mathbf{v}_1 + c_2 \cdot \mathbf{v}_2$ in terms of ϕ as

$$\mathbf{v}(\phi) = \sin\phi \cdot \mathbf{v}_1 + \cos\phi \cdot \mathbf{v}_2. \tag{6.4}$$

This means that we can rewrite the set $\mathcal{S}_2$ as:

$$\mathcal{S}_2 = \{\text{top}_k(\pm(\mathbf{v}(\phi)), \phi \in [0, \pi)\}.$$

Observe that each element of $\mathbf{v}(\phi)$ is a continuous *spectral curve* in ϕ: $[\mathbf{v}(\phi)]_i = [\mathbf{v}_1]_i \sin(\phi) + [\mathbf{v}_2]_i \cos(\phi)$. Consequently, the top/bottom-k supports of $\mathbf{v}(\phi)$, that is $\text{top}_k(\pm\mathbf{v}(\phi))$, are themselves a function of ϕ. How can we find all possible supports?

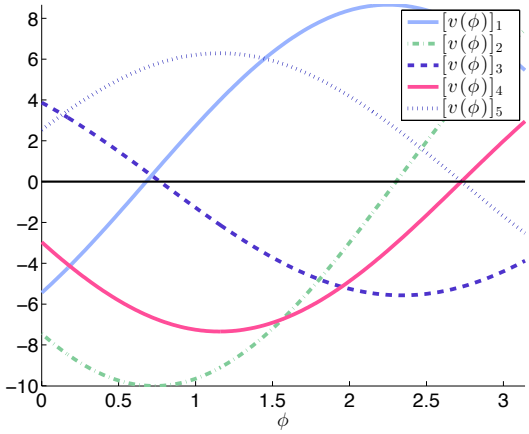


Figure 6.1: A rank $d = 2$ spannogram for $n = 5$ and two random vectors $\mathbf{v}_1, \mathbf{v}_2$. Observe that every two curves intersect in exactly one point. These intersection points define intervals in which a top-k set is invariant.

The Spannogram. In Fig. 6.1, we draw an example plot of five curves $[\mathbf{v}(\phi)]_i$, $i = 1, \ldots, 5$, which we call a spannogram. From the spannogram in Fig. 6.1, we can see that the continuity of these sinusoidal curves implies a "local invariance" property of the top/bottom k supports $\text{top}_k(\pm\mathbf{v}(\phi))$, in a small neighborhood around a fixed ϕ. So, when does a top/bottom-k support change? The index sets $\text{top}_k(\pm\mathbf{v}(\phi))$ change *if and only if* two curves cross, *i.e.*, when the ordering of two elements $[\mathbf{v}(\phi)]_i$,$[\mathbf{v}(\phi)]_j$ changes.

Finding all supports: There are n curves and each pair intersects at exactly one point in the Φ domain[5]. Therefore, there are exactly $\binom{n}{2}$ intersection points. These $\binom{n}{2}$ intersection points define $\binom{n}{2}+1$ intervals. Within an interval the top, or bottom, k supports $\text{top}_k(\pm\mathbf{v}(\phi))$ *remain the same*. Hence, it is now clear that $|\mathcal{S}_2| \leq 2\binom{n}{2} = O(n^2)$.

A way to find all supports in $\mathcal{S}_2$ is to compute the $\mathbf{v}(\phi_{i,j})$ vectors on the intersection points of two curves i, j, and then the supports in the two adjacent intervals of such intersection point. The $\mathbf{v}(\phi_{i,j})$ vector on an intersection point of two curves i and j can be easily computed by first solving a set of linear equations $[\mathbf{v}(\phi_{i,j})]_i = [\mathbf{v}(\phi_{i,j})]_j \Rightarrow (\mathbf{e}_i - \mathbf{e}_j)^T[\mathbf{v}_1 \ \mathbf{v}_2]\mathbf{c}_{i,j} = \mathbf{0}_{2\times 1}$ for the unknown vector $\mathbf{c}_{i,j}$, where $\mathbf{e}_i$ is the i-th column of the $n \times n$ identity matrix, i.e., $\mathbf{c}_{i,j} = \text{nullspace}((\mathbf{e}_i - \mathbf{e}_j)^T[\mathbf{v}_1 \ \mathbf{v}_2])$. Then, we compute $\mathbf{v}(\phi_{i,j}) = [\mathbf{v}_1 \ \mathbf{v}_2]\mathbf{c}_{i,j}$. Further details on breaking ties in $\text{top}_k(\mathbf{v}(\phi_{i,j}))$ can be found in Appendix B.

Computational cost: We have $\binom{n}{2}$ intersection points, where we calculate the top/bottom k supports for each $\mathbf{v}(\phi_{i,j})$. The top/bottom k elements of every $\mathbf{v}(\phi_{i,j})$ can be computed in time $O(n)$ using a partial sorting and selection algorithm [26]. Since we perform this routine a total of $O(\binom{n}{2})$ times, the total complexity of the rank-2 algorithm is $O(n^3)$.

General Rank-d case. The algorithm generalizes to arbitrary dimension d, as it relies on the framework of [10]. We provide the details in Appendix B; its pseudo-code is given as Algorithm 3.

Remark 13. *Observe that the computation of each loop under line 2 of Algorithm 3 can be computed in parallel. This will allow us to parallelize the Spannogram.*

6.3.2 An approximate $\mathcal{S}_d$ in nearly-linear time

In the exact solver, we solve DBkS on $\mathbf{A}_d$ in time $O(n^{d+1})$. Surprisingly, when $\mathbf{A}_d$ has only positive eigenvalues, then we can tightly approximate DBkS on $\mathbf{A}_d$ in nearly linear time.

[5]Here we assume that the curves are in *general position*. This can be always accomplished by infinitesimally perturbing the curves as in [79].

Algorithm 4 `Spannogram`(k,$\mathbf{V}_d$)

$\mathcal{S}_d = \emptyset$
for all $(i_1, \ldots, i_d) \in \{1, \ldots, n\}^d$ and $s \in \{-1, 1\}$ **do**
  $\mathbf{c} = s \cdot \text{nullspace}\left(\begin{bmatrix} [(\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_2,:} \\ \vdots \\ [\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_d,:} \end{bmatrix}\right)$
  $\mathbf{v} = \mathbf{V}_d^T \mathbf{c}$
  $\mathcal{S} = \text{top}_k(\mathbf{v})$
  $\mathcal{T} = \mathcal{S} - \{i_1, \ldots, i_d\}$
  for all $\binom{d}{k-|\mathcal{T}|}$ subsets $\mathcal{J}$ of $(i_1, \ldots, i_d)$ **do**
    $\mathcal{S}_d = \mathcal{S}_d \bigcup (\mathcal{T} \cup \mathcal{J})$
  end for
end for
Output: $\mathcal{S}_d$.

Theorem 8. *Let the d largest eigenvalues of the graph be positive, and let the d-th,$(d+1)$-st largest have constant ratio: $\left|\frac{\lambda_d}{\lambda_{d+1}}\right| \geq C$. Then, we can output, with probability $1-\delta$, a k-subgraph with density $(1-\epsilon)^2 \cdot \mathsf{opt}_d$, in time $O\left(m \cdot \log n + \frac{n}{\epsilon^d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$.*

The main idea is that instead of checking *all* $O(n^d)$ possible k sets in $\mathcal{S}_d$, we can approximately solve the problem by randomly sampling $M = O\left(\epsilon^{-d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$ vectors in the span of $\mathbf{v}_1, \ldots, \mathbf{v}_d$. Our proof is based on the fact that we can "approximate" the surface of the d-dimensional sphere with M randomly sampled vectors from the span of $\mathbf{v}_1, \ldots, \mathbf{v}_d$. This allows us to identify, probability $1-\delta$, near-optimal candidates in $\mathcal{S}_d$. The modified algorithm is very simple and is given below; its analysis can be found in Appendix B.

Algorithm 5 `Spannogram_approx`(k, $\mathbf{V}_d, \Lambda_d$)

for $i = 1 : O\left(\epsilon^{-d} \cdot \log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$ **do**
  $\mathbf{v} = (\Lambda_d^{1/2} \cdot \mathbf{V}_d)^T \cdot \texttt{randn}(d, 1)$
  $\mathcal{S}_d = \mathcal{S}_d \cup \text{top}_k(\mathbf{v}) \cup \text{top}_k(-\mathbf{v})$
end for
Output: $\mathcal{S}_d$.

6.4 Scaling up

In this section, we present the two key scalability features that allow us to scale up to graphs with billions of edges.

6.4.1 Vertex Sparsification

We introduce a very simple and efficient pre-processing step for discarding vertices that are unlikely to appear in a top k set in $\mathcal{S}_d$. This step runs after we compute $\mathbf{A}_d$ and uses the leverage score, $\ell_i = \sum_{j=1}^{d} [\mathbf{V}_d]_{i,j}^2 |\lambda_j|$, of the i-th vertex to decide whether we will discard it or not. We show in Appendix B, that by appropriately setting a threshold, we can guarantee a provable bound on the error introduced. In our experimental results, the above elimination is able to reduce n to approximately $\hat{n} \approx 10 \cdot k$ for a provably small additive error, even for data sets where $n = 10^8$.

6.4.2 MapReduce Implementation

A MapReduce implementation allows scaling out to a large number of compute nodes that can work in parallel. The reader can refer to [11, 69]) for a comprehensive treatment of the MapReduce paradigm. In short, the Hadoop/MapReduce infrastructure stores the input graph as a distributed file spread across multiple machines; it provides a tuple streaming abstraction, where each `map` and `reduce` function receives and emits tuples as $(key, value)$ pairs. The role of the keys is to ensure information aggregation: all the tuples with the same key are processed by the same reducer.

For the spectral decomposition step of our scheme we design a simple implementation of the power method in MapReduce. The details are beyond the scope of this work; high-performance implementations are already available in the literature, e.g. [66]. We instead focus on the novel implementation of the Spannogram.

Our MapReduce implementation of the rank-2 Spannogram is outlined in Algorithm 4. The Mapper is responsible for the duplication and dissemination of the eigenvectors, $\mathbf{V}_2, \mathbf{U}_2 = \mathbf{V}_2 \Lambda_2$, to all reducers. Line 3 emits the j-th row of $\mathbf{V}_2$ and $\mathbf{U}_2$ once for every node i. Since i is used as the key, this ensures that every reducer receives $\mathbf{V}_2, \mathbf{U}_2$ in their entirety.

From the breakdown of the Spannogram in Section 6.3, it is understood that, for the rank-2 case, it suffices to solve a simple system of equations for every pair of nodes. The Reducer for node i receives the full eigenvectors $\mathbf{V}_2, \mathbf{U}_2$ and is responsible for solving the problem for every pair (i, j), where $j > i$. Then, Line 6

emits the best candidate computed at Reducer i. A trivial final step, not outlined here, collects all n^2 candidate sets and keeps the best one as the final solution.

The basic outline in Algorithm 4 comes with heavy communication needs and was chosen here for ease of exposition. The more efficient version that we implement, does not replicate $\mathbf{V}_2, \mathbf{U}_2$ n times. Instead, the number of reducers – say $R = n^\alpha$ – is fine-tuned to the capabilities of the cluster. The mappers emit $\mathbf{V}_2, \mathbf{U}_2$ R times, once for every reducer. Then, reducer r is responsible for solving for node pairs (i, j), where $i \equiv r \pmod R$ and $j > i$. Depending on the performance bottleneck, different choices for α are more appropriate. We divide the construction of the $O(n^2)$ candidate sets in $\mathcal{S}_2$ to $O(n^\alpha)$ reducers and each of them computes $O(n^{2-\alpha})$ candidate subgraphs. The total communication cost for this parallelization scheme is $O(n^{1+\alpha})$: n^α reducers need to have access to the entire $\mathbf{V}_2, \mathbf{U}_2$ that has $2 \cdot 2 \cdot n$ entries. Moreover, the total computation cost for each reducer is $O(n^{3-\alpha})$.

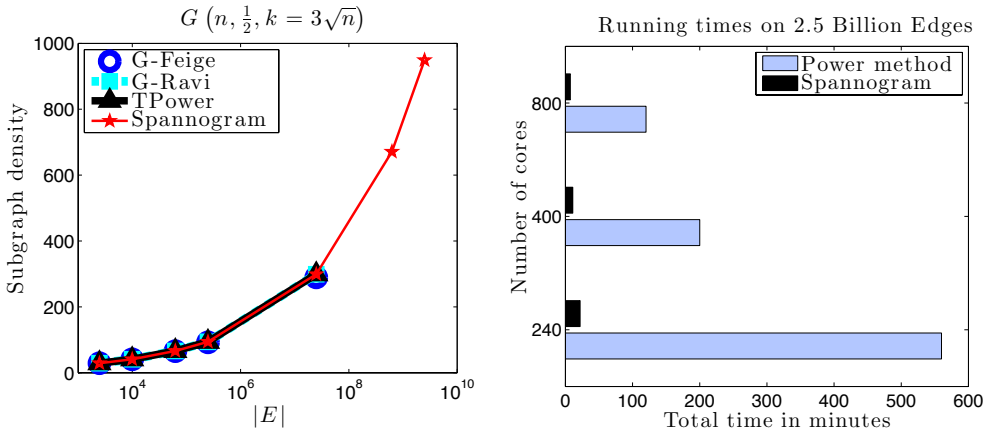
Algorithm 6 `SpannogramMR`$(\mathbf{V}_2, \mathbf{U}_2)$

Map$(\{[\mathbf{V}_2]_{j,:}, [\mathbf{U}_2]_{j,:}, j\})$:
for $i = 1 : n$ **do**
  emit: $\langle i, \{[\mathbf{V}_2]_{j,1}, [\mathbf{V}_2]_{j,2}, [\mathbf{U}_2]_{j,1}, [\mathbf{U}_2]_{j,2}, j\}\rangle$
end for

Reduce$_i(\langle i, \{[\mathbf{V}_2]_{j,1}, [\mathbf{V}_2]_{j,2}, [\mathbf{U}_2]_{j,1}, [\mathbf{U}_2]_{j,2}, j\}\rangle, \forall j)$:
for each $j \geq i + 1$ **do**
  $\mathbf{c} = \text{nullspace}([\mathbf{V}]_{i,:} - [\mathbf{V}]_{j,:})$
  $[\mathsf{den}_j,\ \{\mathcal{X}_j, \mathcal{Y}_j\}] = \max_{|\mathcal{Y}|=k, \mathcal{X} \in \text{top}_k(\pm \mathbf{V}_2 \mathbf{c})} \mathbf{1}_\mathcal{X} \mathbf{V}_2 \mathbf{U}_2^T \mathbf{1}_\mathcal{Y}$
end for
emit: $\left\langle i, \{\mathcal{X}_i, \mathcal{Y}_i\} = \max_j \mathbf{1}_{\mathcal{X}_j} \mathbf{V}_2 \mathbf{U}_2^T \mathbf{1}_{\mathcal{Y}_j} \right\rangle$

6.5 Experimental Evaluation

We experimentally evaluate the performance of our algorithm and compare it to the truncated power method (TPower) of [107], a greedy algorithm by [40] (GFeige) and another greedy algorithm by [84] (GRavi). We performed experiments on synthetic dense subgraphs and also massive real graphs from multiple sources. In all experiments we compare the density of the subgraph obtained by the Spannogram to the density of the output subgraphs given by the other algorithms.



(a) Densities of the recovered subgraph v.s. the expected number of edges.

(b) Running times of the Spannogram and power iteration for two top eigenvectors.

Figure 6.2: Planted clique experiments for random graphs.

Our experiments illustrate three key points: (1) for all tested graphs, our method outperforms – some times significantly – all other algorithms compared; (2) our method is highly scalable, allowing us to solve far larger problem instances; (3) our data-dependent upper bound in many cases provide a certificate of near-optimality, far more accurate and useful, than what *a priori* bounds are able to do.

Planted clique. We first consider the so-called (and now much studied) Planted Clique problem: we seek to find a clique of size k that has been planted in a graph where all other edges are drawn independently with probability $1/2$. We scale our randomized experiments from $n = 100$ up to 10^5. In all cases we set the size of the clique to $k = 3 \cdot \sqrt{n}$ – close to what is believed to be the critical computability threshold. In all our experiments, GRavi, TPower, and the Spannogram successfully recovered the hidden clique. However, as can be seen in Fig. 6.2, the Spannogram algorithm is the only one able to scale up to $n = 10^5$– a massive dense graph with about 2.5 billion edges. The reason is that this graph does not fit in the main memory of one machine and caused all centralized algorithms to crash after several hours. Our MapReduce implementation scales out smoothly, since it splits the problem over multiple smaller problems solved in parallel.

Specifically, we used Amazon Wireless Services' Elastic MapReduce framework [1]. We implemented our `map` and `reduce` functions in Python and used the

MRJob class [3]. For our biggest experiments we used a 100-machine strong cluster, consisting of m1.xlarge AWS instances (a total of 800 cores).

The running times of our experiments over MapReduce are shown in Fig. 2(b). The main bottleneck is the computation of the first two eigenvectors which is performed by repeating the power iteration for few (typically 4) iterations. This step is not the emphasis of this work and has not been optimized. The Spannogram algorithm is significantly faster and the benefits of parallelization are clear since it is CPU intensive.

In principle, the other algorithms could be also implemented over MapReduce, but that requires non-trivial distributed algorithm design. As is well-known, *e.g.*, [69], implementing iterative machine learning algorithms over MapReduce can be a significant task and schemes which perform worse in standard metrics can be highly preferable for this parallel framework. Careful MapReduce algorithmic design is needed especially for dense graphs like the one in the hidden clique problem.

Real Datasets. Next, we demonstrate our method's performance in real datasets and also illustrate the power of our data-dependent bounds. We run experiments on large graphs from different applications and our findings are presented in Fig. 6.3. The figure compares the density achieved by the Spannogram algorithm for rank 1, 2 and 5 to the performance of GFeige, GRavi and TPower. The figure shows that the rank-2 and rank-5 versions of our algorithm, improve – sometimes significantly – over the other techniques. Our novel data-dependent upper-bound shows that our results on these data sets are provably near-optimal.

The experiments are performed for two community graphs (com-LiveJournal and com-DBLP), a web graph (web-NotreDame), and a subset of the Facebook graph. A larger set of experiments is included in the appendix. Note that the largest graph in Figure 6.3 contains no more than 35 million edges; these cases fit in the main memory of a single machine and the running times are presented in the appendix, all performed on a standard Macbook Pro laptop using Matlab. In summary, rank-2 took less than one second for all these graphs while prior work methods took approximately the same time, up to a few seconds. Rank-1 was significantly faster than all other methods in all tested graphs and took fractions of a second. Rank-5 took up to 1000 seconds for the largest graph (LiveJournal).

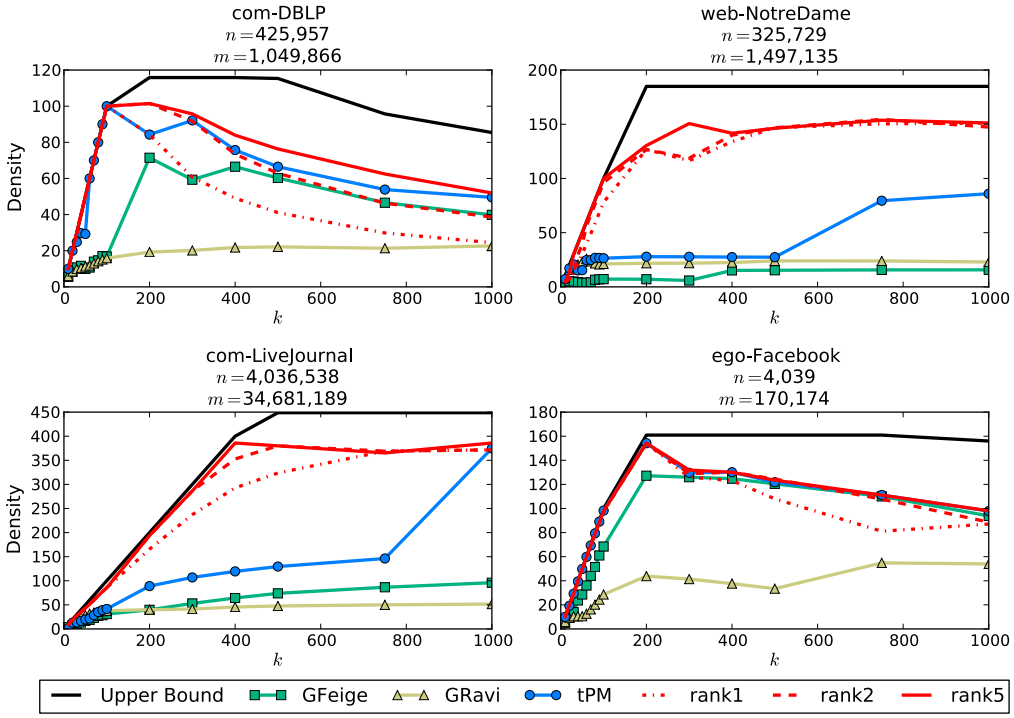


Figure 6.3: Subgraph density vs. subgraph size (k). We compare our DkS Spannogram algorithm with the algorithms from [40] (GFeige), [84] (GRavi), and [107] (tPM). Across all subgraph sizes k, we obtain higher subgraph densities using Spannograms of rank $d = 2$ or 5. We also obtain a *provable data-dependent upper bound* (solid black line) on the objective. This proves that for these data sets, our algorithm is typically within 80% from optimality, for all sizes up to $k = 250$, and indeed for small subgraph sizes we find a clique which is clearly optimal. Further experiments on multiple other data sets are shown in the appendix.

We conclude that our algorithm is an efficient option for finding dense subgraphs. Different rank choices give a tradeoff between accuracy and performance while the parallel nature allows scalability when needed. Further, our theoretical upper-bound can be useful for practitioners investigating dense structures in large graphs.

Appendices

Appendix A

Appendix of Chapter 5

A.1 Nonnegative matrix speed-up

In this section we show that if A has nonnegative entries then we can speed up computations by a factor of 2^{d-1}. The main idea behind this speed-up is that when A has only nonnegative entries, then in our intersection equations found in Subsection B.1 we do not need to check all possible signed combinations of the d curves. In the following we explain this point.

We first note that the Perron-Frobenius theorem [48] grants us the fact that the optimal solution x^* will have nonnegative entries. That is, if A has nonnegative entries, then x^* will also have nonnegative entries. This allows us to pose a redundant nonnegativity constraint on our optimization

$$\max_{x \in \mathbb{S}_k} x^T A x = \max_{x \in \mathbb{S}_k, x \succeq 0} x^T A x. \tag{A.1}$$

Our approximation uses the above constraint to reduce the cardinality of $\mathcal{S}_d$ by a factor of 2^{d-1}. Let us consider for example the rank 1 case:

$$\max_{x \in \mathbb{S}_k, x \succeq 0} \left(v^T x\right)^2 = \max_{x \in \mathbb{S}_k, x \succeq 0} \left(\sum_{i=1}^{n} v_i x_i \right)^2$$

Here,the optimal solution can be again found in time $\mathcal{O}(n \log n)$. First, we sort the elements of v. The optimal support $\mathcal{I}$ the for above problem corresponds to either the top k, or the bottom k unsigned elements of the sorted v. The fact that is important here is that the optimal vector can only have entries of the same sign.[1] The implication of the previous fact is that on our curve intersection points, we can

[1] If there are less than k elements of the same sign in either of the two support sets in $\mathcal{I}_1$, then, and in order to satisfy the sparsity constraint, we can put weight $\epsilon > 0$ on elements with the least amplitude in such set and opposite sign. This will only perturb the objective by a component proportional to ϵ, which can then be driven arbitrarily close to 0, while respecting the sparsity constraint.

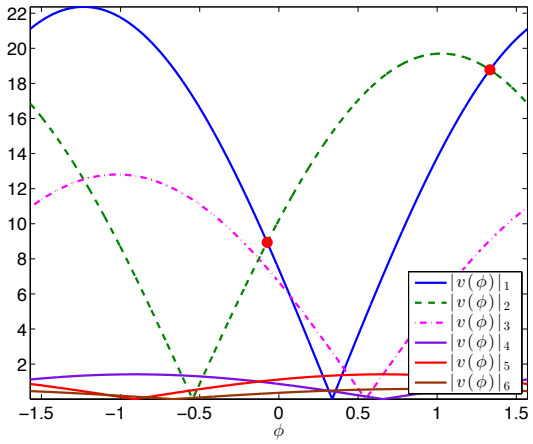
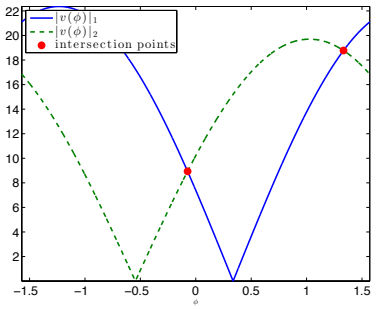


Figure A.1: An example of a spannogram for $n = 6$, $d = 2$. Assume that $k = 1$. Then, the candidate optimal supports are $\mathcal{S}_2 = \{\{1\}, \{2\}\}$, that is either the blue curve ($i = 1$) is the top one, or the green curve ($i = 2$) becomes the top one, depending on the different values of ϕ. Finding the intersection points between these two curves is sufficient to recover these optimal supports. The idea of the elimination is that curves with (maximum) amplitude less than the amplitude of these types of intersection points can be safely discarded. In our example, after considering the blue and green curves and obtaining their intersection points, we can see that all other curves apart from the purple curve can be discarded; their amplitudes are less than the lowest intersection point of the blue and green curves. Our elimination step formalizes this idea.

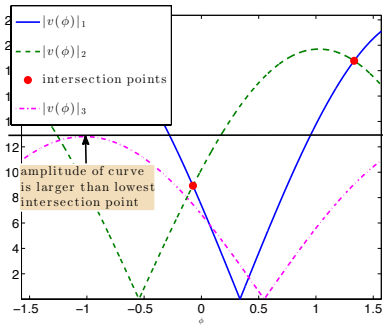
only account for intersections of the sort $[v(\varphi)]_i = [v(\varphi)]_j$. Intersection of the form $[v(\varphi)]_i = -[v(\varphi)]_j$ are not to be considered due to the fact that the locally optimal vector can only have one of the two signs. This means that in the intersection equations that appear in Section B.1, we only have a single sign pattern. This eliminates exactly a factor of 2^{d-1} from the cardinality of the $\mathcal{S}_d$ set.

A.2 Feature Elimination

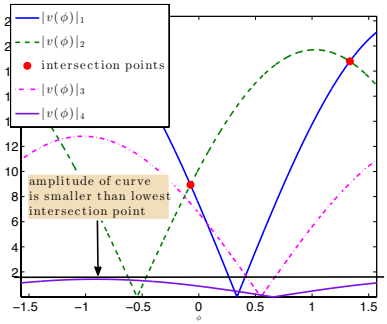
In this section we present our feature elimination algorithm. This step reduces the dimension n of the problem and this reduction in practice is empirically shown to be significantl and allows us to run our algorithm for very large matrices



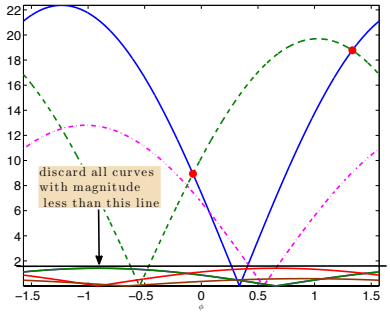
(a) Start with the curves of highest amplitude. Then, find their intersection points (red dots) and put them in the set $\mathcal{P}_1$.



(b) Examine the curve with amplitude that is the largest among the ones not tested yet. If the curve has amplitude less than the minimum intersection point in $\mathcal{P}_1$, discard it. Also, discard all curves with amplitude less than that. If it has amplitude higher than the minimum point in $\mathcal{P}_1$, then compute its intersection points with the curves already examined. For each new intersection point check whether it has $k-1$ curves above it. If yes, add it to $\mathcal{P}_1$. Retest all points in $\mathcal{P}_1$; if there is a point that has more than $k-1$ curves above it, discard it from $\mathcal{P}_1$.



(c) Repeat the same steps. Check if the amplitude of the lowest intersection point is higher than the amplitude of the curve next in the sorted list.



(d) Eventually this process will end by finding a curve with amplitude less than the intersection points in $\mathcal{P}_1$. It will then discard all curves with amplitude less, as shown in the figure above.

Figure A.2: A simple elimination example for $n = 6$, $d = 2$, and $k = 1$.

A. Our elimination algorithm is combinatorial and is based on sequentially checking the rows of V_d, depending on the value of their norm. This step is based again on the spannogram framework used in our approximation algorithm for sparse PCA. In Fig. A.1, we sketch the main idea of our elimination step.

The essentials of the elimination. First note that a locally optimal support set $\mathcal{I}_k(Vc)$ for a fixed c vector, corresponds to the top k elements of v_c. As we mentioned before, all elements of v_c correspond to hypersurfaces $|[v(\varphi)]_i|$ that are functions of the $d-1$ spherical variables in φ. For a fixed $\varphi \in \Phi^{d-1}$, the candidate support set corresponds exactly to the top k (in absolute value) elements in $v_c = v(\varphi)$, or the top k surfaces $|[v(\varphi)]_i|$ for that particular φ. There is a very simple observation here: a surface $|[v(\varphi)]_i|$ belongs to the set of top k surfaces if $|[v(\varphi)]_i|$ is below at most $k-1$ other surfaces on that φ. If it is below k surfaces at that point φ, then $|[v(\varphi)]_i|$ does not belong in the set of k top surfaces.

A second key observation is the following: the only points φ that we need to check are the critical intersection points between d surfaces. For example, we could construct a set $\mathcal{Y}_k$ of all intersection points of all d sets of curves, such that for any point in this set the number of curves above it is at least $k-1$. In other words, $\mathcal{Y}_k$ defines a boundary: if a curve is above this boundary then it may become a top k curve; if not it can never appear in a candidate set. This means that we could test each curve against the points in $\mathcal{Y}_k$ and discard it if its amplitude is less than the amplitudes of all intersection points in $\mathcal{Y}_k$. However, the above elimination technique implies that we would need to calculate all intersection points on the n surfaces. Our goal is to use the above idea by serially checking one by one the intersection points of high amplitude curves.

Elimination algorithm description. We use the above ideas, to build our elimination algorithm. We first compute the norms of each row $\|[V_d]_{:,i}\|_2$ of V_d. This norm corresponds to the amplitude of $[v(\varphi)]_i$. Then, we sort all n rows according to their norms. We first start with the $k+d$ rows of V_d (i.e., surfaces) of highest norm (i.e., amplitude) and compute their $\binom{k+d}{d}$ intersection points. For each intersection point, say ϕ, we compute the number of $|[v(\varphi)]_i|$ surfaces above it. If there are less than $k-1$ surfaces above an intersection point, then this means that such point is a potential intersection point where a new curve enters a local top k set. We keep all these points in a set $\mathcal{P}_k$.

We then move to the $(k+d+1)$-st surface of highest amplitude; we test it against the minimum amplitude point in $\mathcal{P}_k$. If the amplitude of the $(k+d+1)$-st surface is less than the minimum amplitude point in $\mathcal{P}_k$, then we can safely eliminate this surface (i.e., this row of V_d), and all surfaces with maximum amplitude smaller than that (i.e., all rows of V_d with norm smaller than the row of interest). If its amplitude is larger than the amplitude of this point, then we compute the new set of $\binom{k+d+1}{d}$ intersection points obtained by adding this new surface. We check if some of these can be added in $\mathcal{P}_k$, using the test of whether there are at most $k-1$ curves above each point. We need also re-check all previous points in $\mathcal{P}_k$, since some may no longer be eligible to be in the set; if some are not, then we delete them from the set $\mathcal{P}_k$. We then move on the next row of V_d, and continue this process until we reach a row with norm less than the minimum amplitude of the points in $\mathcal{P}_k$.

A pseudo-code for our feature elimination algorithm can be found as Algorithm 1. In Fig. A.2, we give an example of how our elimination works.

Algorithm 7 Elimination Algorithm.

Input: k, p, $V_d = \left[\sqrt{\lambda_1}v_1 \dots \sqrt{\lambda_1}v_1\right]$
Initialize $\mathcal{P}_k \leftarrow \emptyset$
Sort the rows of V_d in descending order according to their norms $\|e_i^T V_d\|$.
$\tilde{n} \leftarrow k + d + 1$.
$\tilde{V} \leftarrow V_{1:\tilde{n},:}$.
for all $\binom{\tilde{n}}{d}$ subsets $(i_1, \dots, i_d)$ from $\{1, \dots, \tilde{n}\}$ **do**
  for all sequences $(b_1, \dots, b_{d-1}) \in \mathcal{B}$ **do**
    $c \leftarrow$ nullspace $\left(\begin{bmatrix} e_{i_1}^T - b_1 \cdot e_{i_2}^T \\ \vdots \\ e_{i_1}^T - b_{d-1} \cdot e_{i_d}^T \end{bmatrix} V_d \right)$
    if there are $k-1$ elements of $|v_c|$ greater than $|e_1 V_d c|$ **then**
      $\mathcal{P}_k \leftarrow \mathcal{P}_k \cup \{|e_1 V_d c|\}$
    end if
    if $\|V_{\tilde{n}+1,:}\| < \min\{x \in \mathcal{P}_k\}$ **then**
      STOP ITERATIONS.
    end if
    $\tilde{n} \leftarrow \tilde{n} + 1$
    $\tilde{V} \leftarrow V_{1:\tilde{n},:}$
    for each element x in $\mathcal{P}_k$ **do**
      check the elements $|v_c|$ greater than x
      if there are more than $k-1$ **then**
        discard it
      end if
    end for
  end for
end for
Output: $\tilde{A}_d = \tilde{V}_d \tilde{V}_d^T$, where $\tilde{V}_d$ comprises of the first $\tilde{n}$ rows of V_d of highest norm.

A.3 Approximation Guarantees

In this section, we prove the approximation guarantees for our algorithm. Let us define two quantities, namely

$$\text{OPT} = \max_{x \in \mathbb{S}_k} x^T A x \;\text{ and }\; \text{OPT}_d = \max_{x \in \mathbb{S}_k} x^T A_d x,$$

which correspond to the optimal values of the initial maximization under the full-rank matrix A and its rank-d approximation A_d, respectively. Then, we establish the following lemma.

Lemma 14. *Our approximation factor is lower bounded as*

$$\rho_d = \frac{\max_{\mathcal{I} \in \mathcal{S}_d} \lambda(A_{\mathcal{I}})}{\lambda_1^{(k)}} \geq \frac{OPT_d}{OPT}. \tag{A.2}$$

Proof. The first technical fact that we use is that an optimizer vector x_d for A_d (i.e., the one with the maximum performance for A_d), can achieve at least the same performance for A, i.e., $x_d^T A x_d \geq x_d^T A_d x_d$. The proof is straightforward: since A is PSD, each quadratic form inside the sum $\sum_{i=1}^{n} \lambda_i x^T v_i v_i^T x$ is a positive number. Hence, $\sum_{i=1}^{n} \lambda_i x^T v_i v_i^T x \geq \sum_{i=1}^{d} \lambda_i x^T v_i v_i^T x$, for any vector x and any d.

The second technical fact is that if we are given a vector x_d with nonzero support $\mathcal{I}$, then calculating q_d, the principal eigenvector of $A_\mathcal{I}$, results in a solution for A with better performance compared to x_d. To show that, we first rewrite x_d as $x_d = P_\mathcal{I} x_d$, where $P_\mathcal{I}$ is an $n \times n$ matrix that has 1s on the diagonal elements that multiply the nonzero support of x_d and has 0s elsewhere. Then, we have

$$\begin{aligned} \text{OPT}_d \leq x_d^T A x_d = x_d^T P_\mathcal{I} A P_\mathcal{I} x_d = x_d^T A_\mathcal{I} x_d \\ \leq \max_{\|x\|_2=1} x^T A_\mathcal{I} x = q_d^T A_\mathcal{I} q_d = q_d^T A q_d. \end{aligned} \tag{A.3}$$

Using the above fact for all sets $\mathcal{I} \in \mathcal{S}_d$, we obtain that $\max_{\mathcal{I} \in \mathcal{S}_d} \lambda(A_\mathcal{I}) \geq \text{OPT}_d$, which proves our lower bound. □

Sparse spectral ratio. A basic quantity that is important in our approximation ratio as we see in the following, is what we define as the *sparse spectral ratio*, which is equal to $\lambda_{d+1}/\lambda_1^{(k)}$. This ratio will be shown to be directly related to the (non-sparse) spectrum of A.

Here we prove the the following lemma.

Lemma 15. *Our approximation ratio is lower bounded as follows.*

$$\rho_d \geq 1 - \frac{\lambda_{d+1}}{\lambda_1^{(k)}}. \tag{A.4}$$

Proof. We first decompose the quadratic form in (5.1) in two parts

$$x^T A x = x^T \left(\sum_{i=1}^{n} \lambda_i v_i v_i^T \right) x = x^T A_d x + x^T A_{d^c} x \tag{A.5}$$

where $A_{d^c} = A - A_d = \sum_{i=d+1}^{n} \lambda_i v_i v_i^T$. Then, we take maximizations on both parts of (A.5) over our feasible set of vectors with unity ℓ_2-norm and cardinality k and

obtain

$$\begin{aligned}
&\max_{x\in\mathbb{S}_k} x^T A x = \max_{x\in\mathbb{S}_k} \left(x^T A_d x + x^T A_{d^c} x\right)\\
\overset{(i)}{\Leftrightarrow} &\max_{x\in\mathbb{S}_k} x^T A x \leq \max_{x\in\mathbb{S}_k} x^T A_{d^c} x + \max_{x\in\mathbb{S}_k} x^T A_{d^c} x\\
\Leftrightarrow &\text{OPT} \leq \text{OPT}_d + \max_{x\in\mathbb{S}_k} x^T A_{d^c} x\\
\overset{(ii)}{\Leftrightarrow} &\text{OPT} \leq \text{OPT}_d + \max_{\|x\|_2=1} x^T A_{d^c} x\\
\overset{(iii)}{\Leftrightarrow} &\text{OPT} \leq \text{OPT}_d + \lambda_{d+1},
\end{aligned} \tag{A.6}$$

where (i) comes from the fact that the maximum of the sum of two quantities is always upper bounded by the sum of their maximum possible values, (ii) is due to the fact that we lift the ℓ_0 constraint on the optimizing vector x, and (iii) is due to the fact that the largest eigenvalue of $A - A_d$ is equal to λ_{d+1}. Moreover, due to the fact that $\text{OPT} \geq \text{OPT}_d$, we have

$$\text{OPT} - \lambda_{d+1} \leq \text{OPT}_d \leq \text{OPT}. \tag{A.7}$$

Dividing both the terms of (A.7) with OPT yields

$$1 - \frac{\lambda_{d+1}}{\lambda_1^{(k)}} = 1 - \frac{\lambda_{d+1}}{\text{OPT}} \leq \frac{\text{OPT}_d}{\text{OPT}} \leq \rho_d \leq 1. \tag{A.8}$$

□

Lower-bounding $\lambda_1^{(k)}$. We will now give two lower-bounds on OPT.

Lemma 16. *The sparse eigenvalue of A is lower bounded as*

$$\lambda_1^{(k)} \geq \max\left\{\frac{k}{n}\lambda_1, \lambda_1^{(1)}\right\}. \tag{A.9}$$

Proof. The second bound is straightforward: if we assume the feasible solution $e_{\max}$, being the column of the identity matrix which has a 1 in the same position as the maximum diagonal element of A, then we get

$$\text{OPT} \geq e_{\max}^T C e_{\max}^T = \max_{i=1,\ldots,n} A_{ii} = \lambda_1^{(1)}. \tag{A.10}$$

The first bound for OPT will be obtained by examining the rank-1 optimal solution on A_1. Observe that

$$\begin{aligned} \text{OPT} \geq \text{OPT}_1 &= \max_{x \in \mathbb{S}_k} x^T A_1 x \\ &= \lambda_1 \max_{x \in \mathbb{S}_k} (v_1^T x)^2. \end{aligned} \tag{A.11}$$

Both v_1 and x have unit norm; this means that $(v_1^T x)^2 \leq 1$. The optimal solution for this problem is to allocate all k nonzero elements of x on $\mathcal{I}_k(v_1)$: the top k absolute elements of v_1. An optimal solution vector, will give a metric of $(v_1^T x)^2 = \|[v_1]_{\mathcal{I}_k(v_1)}\|_2^2$. The norm of the k largest elements of v_1 is then at least equal to $\frac{k}{n}$ times the norm of v_1. Therefore, we have

$$\text{OPT} \geq \max \left\{ \frac{k}{n} \lambda_1, \lambda_1^{(1)} \right\}. \tag{A.12}$$

□

The above lemmata can be combined to establish Theorem 1.

A.4 Resolving singularities

In our algorithmic developments, we have made an assumption on the curves studied, i.e., on the rows of the V_d matrix. This assumption was made so that tie-breaking cases are evaded, where more than d curves intersect in a single point in the d dimensional space Φ^d. Such a singularity is possible even for full-rank matrices V_d and can produce enumerating issues in the generation of locally optimal candidate vectors that are obtained through the intersection equations:

$$\begin{bmatrix} e_{i_1}^T - b_1 e_{i_2}^T \\ \vdots \\ e_{i_1}^T - b_{d-1} e_{i_d}^T \end{bmatrix}_{d-1 \times n} V_d c = 0_{d-1 \times 1}. \tag{A.13}$$

The above requirement can be formalized as: no system of equations of the following form has a nontrivial (i.e., nonzero) solution

$$\begin{bmatrix} e_{i_1}^T - b_1 e_{i_2}^T \\ \vdots \\ e_{i_1}^T - b_{d-1} e_{i_d}^T \\ e_{i_1}^T - b_{d-1} e_{i_{d+1}}^T \end{bmatrix}_{d \times n} V_d c \neq 0_{d \times 1} \tag{A.14}$$

for all $c \neq 0$ and all possible $d+1$ row indices $i_1, \ldots, i_{d+1}$ (where two indices cannot be the same). We show here that the above issues can be avoided by slightly perturbing the matrix V_d. We will also show that this perturbation is not changing the approximation guarantees of our scheme, guaranteed that it is sufficiently small. We can thus rewrite our requirement as a full-rank assumption on the following matrices

$$\text{rank}\left(\begin{bmatrix} e_{i_1}^T - b_1 e_{i_2}^T \\ \vdots \\ e_{i_1}^T - b_{d-1} e_{i_d}^T \\ e_{i_1}^T - b_{d-1} e_{i_{d+1}}^T \end{bmatrix}_{d \times n} V_d\right) = d \tag{A.15}$$

for all $i_1 \neq i_2 \neq \ldots \neq i_d$. Observe that we can rewrite the above matrix as

$$\begin{bmatrix} e_{i_1}^T - b_1 e_{i_2}^T \\ \vdots \\ e_{i_1}^T - b_{d-1} e_{i_d}^T \\ e_{i_1}^T - b_{d-1} e_{i_{d+1}}^T \end{bmatrix}_{d \times n} V_d = \begin{bmatrix} [V_d]_{i_1,:} - b_1 [V_d]_{i_2,:} \\ \vdots \\ [V_d]_{i_1,:} - b_1 [V_d]_{i_d,:} \\ [V_d]_{i_1,:} - b_1 [V_d]_{i_{d+1},:} \end{bmatrix}$$

$$= \begin{bmatrix} [V_d]_{i_1,:} \\ \vdots \\ [V_d]_{i_1,:} \\ [V_d]_{i_1,:} \end{bmatrix} - \begin{bmatrix} b_1 [V_d]_{i_2,:} \\ \vdots \\ b_1 [V_d]_{i_d,:} \\ b_1 [V_d]_{i_{d+1},:} \end{bmatrix} = R_{i_1} + G_{i_2,\ldots,i_{d+1}}$$

where R_{i_1} is a rank-1 matrix. Observe that the rank of the above matrix depends on the ranks of both of its components and how the two subspaces interact. It should not be hard to see that we can add a $d \times d$ random matrix $\Delta = [\delta_1 \delta_2 \ldots \delta_d]$ to the above matrix, so that $R_{i_1} + G_{i_2,\ldots,i_{d+1}} + \Delta$ is full-rank with probability 1.

Let E_d be an $n \times d$ matrix with entries that are uniformly distributed and bounded as $|E_{i,j}| \leq \epsilon$. Instead of working on V_d we will work on the perturbed matrix $\tilde{V}_d = V_d + E_d$. Then, observe that for any matrix of the previous form $R_{i_1} + G_{i_2,\ldots,i_{d+1}}$ we now have $R_{i_1} + G_{i_2,\ldots,i_{d+1}} + [E_d]_{i_1,:} \otimes 1_{d \times 1} + E_{i_2,\ldots,i_{d+1}}$, where

$$E_{i_2,\ldots,i_d} = \begin{bmatrix} [E_d]_{i_2,:} \\ [E_d]_{i_3,:} \\ \vdots \\ [E_d]_{i_{d+1},:} \end{bmatrix}. \tag{A.16}$$

Conditioned on the randomness of $[E_d]_{i_1,:}$, the matrix $R_{i_1} + G_{i_2,\ldots,i_{d+1}} + [E_d]_{i_1,:} \otimes 1_{d \times 1} + E_{i_2,\ldots,i_{d+1}}$ is full rank. Then due to the fact that there are d random variables

in $[E_d]_{i_1,:}$ and d^2 random variable in $R_{i_1} + G_{i_2,\ldots,i_{d+1}} + [E_d]_{i_1,:} \otimes 1_{d\times 1} + E_{i_2,\ldots,i_{d+1}}$, the latter matrix will be full-rank with probability 1 using a union bounding argument. This means that all $\binom{n}{d}$ submatrices of $\tilde{V}_d$ will be full rank, hence obtaining the property that no $d+1$ curves intersect in a single point in Φ^d.

Now we will show that this small perturbation does not change our metric of interest significantly. The following holds for any unit norm vector x

$$\begin{aligned} x^T(V_d + E_d)(V_d + E_d)^T x &= x^T V_d V_d^T x + x^T E_d E_d^T x + 2x^T V_d E_d^T x \\ &\geq x^T V_d V_d^T x + 2x^T V_d E_d^T x \\ &\geq x^T V_d V_d^T x - 2\|V_d^T x\| \cdot \|E_d^T x\| \\ &\geq x^T V_d V_d^T x - 2\sqrt{\lambda_1 \cdot \lambda_1(E_d E_d^T)} \end{aligned}$$

and

$$\begin{aligned} \|(V_d + E_d)^T x\|^2 &\leq \|V_d^T x\|^2 + 2\|E_d x\|\|V_d^T x\| + \|E_d^T x\|^2 \\ &\leq \|V_d^T x\|^2 + 2\sqrt{\lambda_1 \cdot \lambda_1(E_d E_d^T)} + \lambda_1(E_d E_d^T) \\ &\leq \|V_d^T x\|^2 + 3\sqrt{\lambda_1 \cdot \lambda_1(E_d E_d^T)}. \end{aligned}$$

Combining the above we obtain the following bound

$$x^T V_d V_d^T x - 3\sqrt{\lambda_1 \cdot \lambda_1(E_d E_d^T)} \leq \|(V_d + E_d)^T x\|^2 \leq \|V_d^T x\|^2 + 3\sqrt{\lambda_1 \cdot \lambda_1(E_d E_d^T)}.$$

By the above, we can appropriately pick ϵ such that $3\sqrt{\lambda_1 \cdot \lambda_1(E_d E_d^T)} = o(1)$. An easy way to get a bound on is via the Gershgorin circle theorem [48], which yields $\lambda_1(E_d E_d^T) < nd \cdot \epsilon^2$. Hence, an $\epsilon < \frac{1}{\sqrt{\lambda_1 nd} \log n}$ works for our purpose.

To summarize, in the above we show that there is an easy way to avoid singularities in our problem. Instead of solving the original rank-d problem on V_d, we can instead solve it on $V_d + E_d$, with an E_d random matrix with sufficiently small entries. This slight perturbation only incurs an error of at most $\frac{1}{\log n}$ in the objective, which asymptotically becomes zero as n increases.

A.5 Twitter data-set description

In Table A.1, we give an overview of our Twitter data set.

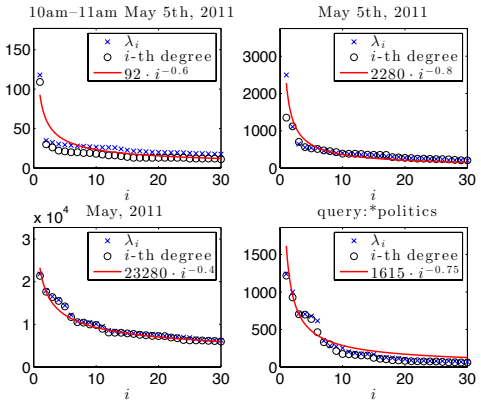
Data-set Specifications	
Geography	mostly Greece
time-window	January 1-August 20, 2011
unique user IDs	$\sim$ 120k
size	$\sim$10 million entries
tweets/month	~ 1.5 million
tweets/day	$\sim$ 70k
tweets/hour	$\sim$ 3k
words/tweet	~ 5
character limit/tweet	140

Table A.1: Data-set specifications.

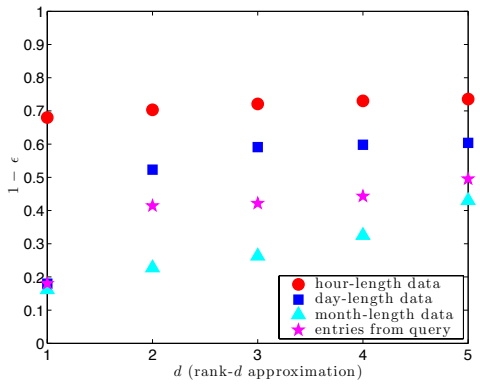
A.5.1 Power Laws

In this subsection we provide empirical evidence that our tested data-sets exhibit a power law decay on their spectrum We report these observations as a proof of concept for our approximation guarantees. Based on the spectrum of some subsets of our data-set, we provide the exact approximation guarantees derived using our bounds.

In Fig. A.3, we plot the best fit power law for the spectrum and degrees with data-set parameters given on the figures. The plots that we provide are for hour-length, day-length, and month-length analysis, and subsets of our data set based on a specific query. We observe that for all these subsets of our data set, the spectrum indeed follows a power-law. An interesting observation is that a very similar power law is followed by the degrees of the terms in the data set. This finding is compatible to the generative models and analysis of [25, 70]. The rough overview is that eigenvalues of A can be well approximated using the diagonal elements of A. In the same figure, we show how our approximation guarantees that based on the spectrum of A scales with d, for the various data-sets tested. We only plot for d up to 5, since for any larger d our algorithm becomes impractical for moderately large small data sets.



(a) The plots that we provide are for hour-length, day-length, and month-length analysis, and subsets of our data set based on a specific query.



(b) Approximation Guarantees: we show how the approximation guarantees for these specific subsets of our data set scales with d.

Figure A.3: Power laws and approximation guarantees

Appendix B

Appendix for Chapter 6

B.1 Proof of Lemma 4: Building the set $\mathcal{S}_d$ for arbitrary d-dimensional subspaces

In our general case, we solve DBkS on

$$\mathbf{A}_d = \mathbf{V}_d \mathbf{U}_d^T = \sum_{i=1}^{d} \mathbf{v}_i \mathbf{u}_i^T$$

where

$$\mathbf{V}_d = [v_1 \ \ldots \ v_d] \text{ and } \mathbf{U}_d = [\lambda_1 \cdot v_1 \ \ldots \ \lambda_d \cdot v_d] .$$

Solving the problem on $\mathbf{A}_d$ is equivalent to answering the following combinatorial question:

"how many different top-k supports are there in a d-dimensional subspace:
$top_k(c_1 \cdot \mathbf{v}_1 + \ldots + c_d \cdot \mathbf{v}_d)$*?"*

This question was answered in [10], and in the following we will provide the analysis for completeness. For more details please refer to [10]. Let us define $d-1$ auxiliary angles $\phi_1, \ldots, \phi_{d-1} \in \Phi = [0, \ \pi)$ and we rewrite the coefficients $c_1, \ldots, c_d$ as

$$\mathbf{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_d \end{bmatrix} = \begin{bmatrix} \sin\phi_1 \\ \cos\phi_1 \sin\phi_2 \\ \vdots \\ \cos\phi_1 \cos\phi_2 \ldots \sin\phi_{d-1} \\ \cos\phi_1 \cos\phi_2 \ldots \cos\phi_{d-1} \end{bmatrix}.$$

Clearly we can express every vector in the span of $\mathbf{V}_d$ as a linear combination $c_1 \cdot \mathbf{v}_1 + \ldots + c_d \cdot \mathbf{v}_d$ in terms of ϕ:

$$\mathbf{v}(\phi_1, \ldots, \phi_{d-1}) = (\sin\phi_1) \cdot \mathbf{v}_1 + (\cos\phi_1 \sin\phi_2) \cdot \mathbf{v}_2 + \ldots + (\cos\phi_1 \cos\phi_2 \ldots \cos\phi_{d-1}) \cdot \mathbf{v}_d. \tag{B.1}$$

For notation simplicity let us define a vector that contains all $d-1$ auxiliary phase variables

$$\varphi = [\phi_1, \ldots, \phi_{d-1}].$$

We can use the above derivations to rewrite the set $\mathcal{S}_d$ that contains all top k coordinates in the span of $\mathbf{V}_d$ as:

$$\begin{aligned}\mathcal{S}_d &= \{\text{top}_k(c_1 \cdot \mathbf{v}_1 + \ldots + c_d \cdot \mathbf{v}_d) : c_1, \ldots, c_d \in \mathbb{R}\} \\ &= \{\text{top}_k(\pm(\mathbf{v}(\varphi)) : \varphi \in \Phi^{d-1}\} \\ &= \{\text{top}_k(\pm((\sin\phi_1) \cdot \mathbf{v}_1 + \cdot\mathbf{v}_2 + \ldots + (\cos\phi_1 \cos\phi_2 \ldots \cos\phi_{d-1}) \cdot \mathbf{v}_d), \varphi \in \Phi^{d-1}\}.\end{aligned}$$

Observe again that each element of $\mathbf{v}(\varphi)$ is a continuous *spectral curve* in the $d-1$ auxiliary variables:

$$[\mathbf{v}(\varphi)]_i = (\sin\phi_1) \cdot [\mathbf{v}_1]_i + (\cos\phi_1 \sin\phi_2) \cdot [\mathbf{v}_2]_i + \ldots + (\cos\phi_1 \cos\phi_2 \ldots \cos\phi_{d-1}) \cdot [\mathbf{v}_d]_i.$$

Consequently, the top/bottom-k supports of $\mathbf{v}(\varphi)$ (*i.e.*, $\text{top}_k(\pm\mathbf{v}(\varphi))$) are themselves a function of the $d-1$ variables in φ. How can we find all possible supports?

Remark 14. *In our general problem we wish to find all top and bottom k coordinates that appear in a d-dimensional subspace. In the following discussion, for simplicity we handle the top k coordinates problem. Finding the bottom k trivially follows, by just checking the smallest k coordinates of each vector $c_1 \cdot \mathbf{v}_1 + \ldots + c_d \cdot \mathbf{v}_d$ that we construct using our algorithm.*

B.1.1 Ranking regions for a single coordinate $[\mathbf{v}(\varphi)]_i$

We now show that for each single coordinate $[\mathbf{v}(\varphi)]_i$, we can partition Φ^{d-1} in regions, wherein the ith coordinate $[\mathbf{v}(\varphi)]_i$ retains the same ranking relative to the other $n-1$ coordinates in the vector $\mathbf{v}(\varphi)$.

Let us first consider for simplicity $[\mathbf{v}(\varphi)]_1$. We aim to find all values of φ where $[\mathbf{v}(\varphi)]_1$ is in one of the the k largest coordinates of $\mathbf{v}(\varphi)$. We observe that this region can be characterized by using n boundary tests:

$$\begin{aligned}[\mathbf{v}(\varphi)]_1 &\gtrless [\mathbf{v}(\varphi)]_2 \\ [\mathbf{v}(\varphi)]_1 &\gtrless [\mathbf{v}(\varphi)]_3 \\ &\vdots \\ [\mathbf{v}(\varphi)]_1 &\gtrless [\mathbf{v}(\varphi)]_n\end{aligned}$$

Each of the above boundary tests defines a bounding curve that partitions the Φ^{d-1} domain. We refer to this bounding curve as $\mathcal{B}_{1,j}(\varphi) : \Phi^{d-1} \mapsto \Phi^{d-2}$. A $\mathcal{B}_{1,j}(\varphi)$ curve partitions Φ and defines two regions of φ angles:

$$\mathcal{R}_{1>j} = \{\varphi \in \Phi^{d-1} : [\mathbf{v}(\varphi)]_1 > [\mathbf{v}(\varphi)]_j\} \text{ and } \mathcal{R}_{1\leq j} = \{\varphi \in \Phi^{d-1} : [\mathbf{v}(\varphi)]_1 \leq [\mathbf{v}(\varphi)]_j\} \tag{B.2}$$

such that $\mathcal{R}_{1>j} \cup \mathcal{R}_{1\leq j} = \Phi^{d-1}$.

Observe that these $n-1$ curves $\mathcal{B}_{1,1}(\varphi), \ldots, \mathcal{B}_{1,n}(\varphi)$ partition Φ in disjoint *cells*, $\mathcal{C}_1^1, \ldots, \mathcal{C}_T^1$, such that

$$\bigcup_{i=1}^{T} \mathcal{C}_i^1 = \Phi^{d-1}.$$

Within each cell $\mathcal{C}_i^1$, the first coordinate $[\mathbf{v}(\varphi)]_1$ retains a fixed ranking relative to the rest of the elements in $\mathbf{v}(\varphi)$, e.g., for a specific cell it might be the largest element, and in another cell it might be the 10th smallest, etc. This happens because for all values of φ in a single cell, the respective ordering $[\mathbf{v}(\varphi)]_1 \gtrless [\mathbf{v}(\varphi)]_2, \ \ldots, \ [\mathbf{v}(\varphi)]_1 \gtrless [\mathbf{v}(\varphi)]_n$ remains the same.

If we have access to a single point, say φ_0, that belongs to a specific cell, say $\mathcal{C}_j^1$, then we can calculate $[\mathbf{v}(\varphi_0)]$ and find the ranking of the first coordinate $[\mathbf{v}(\varphi)]_1$, that remains invariant for all $\varphi \in \mathcal{C}_j^1$. Hence, if we visit all these cells, then we can find all possible rankings that the first coordinate $[\mathbf{v}(\varphi)]_1$ takes in the d-dimensional span of $\mathbf{v}_1, \ldots, \mathbf{v}_d$. In the following subsections, we show that the number of these cells is bounded by $T \leq 2^d \binom{n-1}{d-1}$.

Observe that each bounding curve $\mathcal{B}_{1,i}(\varphi)$ has a one-to-one correspondence to an equation $[\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_j$, which is linear in $\mathbf{c}$:

$$[\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_j \Rightarrow \mathbf{e}_1^T \mathbf{V}_d \mathbf{c} - \mathbf{e}_i^T \mathbf{V}_d \mathbf{c} = 0 \Rightarrow (\mathbf{e}_1 - \mathbf{e}_j)^T \mathbf{V}_d \mathbf{c} = 0. \tag{B.3}$$

Due to their linear characterization with respect to $\mathbf{c}$, it is easy to see that each $(d-1)$-tuple of bounding curves intersects on a single point in Φ^{d-1}:[1]

$$\begin{array}{c} [\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_{i_1} \\ [\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_{i_2} \\ \vdots \\ [\mathbf{v}(\varphi)]_1 = [\mathbf{v}(\varphi)]_{i_{d-1}} \end{array} \begin{array}{c} \\ \\ \\ \\ \end{array} \begin{array}{c} (\mathbf{e}_1 - \mathbf{e}_{i_1})^T \mathbf{V}_d \mathbf{c} = 0 \\ (\mathbf{e}_1 - \mathbf{e}_{i_2})^T \mathbf{V}_d \mathbf{c} = 0 \\ \vdots \\ (\mathbf{e}_1 - \mathbf{e}_{i_{d-1}})^T \mathbf{V}_d \mathbf{c} = 0 \end{array} \Rightarrow \begin{bmatrix} (\mathbf{e}_1 - \mathbf{e}_{i_1})^T \\ (\mathbf{e}_1 - \mathbf{e}_{i_2})^T \\ \vdots \\ (\mathbf{e}_1 - \mathbf{e}_{i_{d-1}})^T \end{bmatrix} \mathbf{V}_d \mathbf{c} = \mathbf{0}_{(d-1)\times 1}.$$

[1] as a matter of fact, due to the sign ambiguity of the solution, this corresponds to two intersection points. However, the following discussion omits this technical detail for simplicity.

Let us denote the solution of the above linear inverse problem as $\mathbf{c}_{1,i_1,\ldots,i_{d-1}}$. We refer to $\mathbf{c}_{1,i_1,\ldots,i_{d-1}}$ as an intersection vector. For each intersection vector $\mathbf{c}_{1,i_1,\ldots,i_{d-1}}$, we can compute its polar expression and solve for the angles φ that generate it. These $d-1$ input angles correspond exactly to the intersection point of $d-1$ curves specified by the above $d-1$ equations. We denote these $d-1$ angles that generate $\mathbf{c}_{1,i_1,\ldots,i_{d-1}}$, as $\varphi_{1,i_1,\ldots,i_{d-1}}$ which we refer to as the intersection point of the $d-1$ curves $\mathcal{B}_{1,i_1}(\varphi),\ldots,\mathcal{B}_{1,i_{d-1}}(\varphi)$.

Since, the $\varphi_{1,i_1,\ldots,i_{d-1}}$ intersection points are defined for every $d-1$ curves, the total number of intersection points is $\binom{n-1}{d-1}$. In the following subsections, we show how we can visit all cells by just examining these intersection points.

We proceed to show that if we visit the adjacent cells of the intersection points defined for all coordinates, then we can find all top-k supports in the span of $\mathbf{V}_d$.

B.1.2 Visiting all cells = finding all top k supports

Our goal is to find all top-k supports that can appear in the span of $\mathbf{V}_d$. To do so, it is sufficient to visit the cells where $[\mathbf{v}(\varphi)]_1$ is the k-th largest coordinate, then the cells where $[\mathbf{v}(\varphi)]_2$ is the k-largest, and so on. Within such cells, one coordinate (say $[\mathbf{v}(\varphi)]_i$) remains always the k-th largest, while the identities of the bottom $n-k$ coordinates remain the same. This means that in such a cell, we have that

$$[\mathbf{v}(\varphi)]_i \geq [\mathbf{v}(\varphi)]_{j_1},\ldots,[\mathbf{v}(\varphi)]_i \geq [\mathbf{v}(\varphi)]_{j_{n-k}}$$

for all φ in that cell and some scecific $n-k$ other coordinates indexed by $j_1,\ldots,j_{n-k}$. Hence, although the sorting of the top $k-1$ elements might change in that cell (i.e., the first might become the second largest, and vice versa), the coordinates that participate in the top $k-1$ support will be the same, while at the same time the k-th largest will be $[\mathbf{v}(\varphi)]_i$.

Hence, for each coordinate $[\mathbf{v}(\varphi)]_i$, we need to visit the cells wherein it is the k-th largest. We do this by examining *all* cells wherein $[\mathbf{v}(\varphi)]_i$ retains a fixed ranking. Visiting all these cells (T for each coordinate), is possible by visiting all $n\cdot\binom{n-1}{d-1}$ intersection points of $\mathcal{B}_{i,j}(\varphi)$ curves as defined earlier. Since we know that each cell is adjacent to at least 1 intersection point, then at each of these points we visit all adjacent cells. For each cell that we visit, we compute the support of the

largest k coordinates of a vector $\mathbf{v}(\varphi_0)$ with a φ_0 that lies in that cell. We include this top k index set in $\mathcal{S}_d$ and carry the same procedure for all cells. Since we visit all coordinates and all their adjacent cells, this means that we visit all cells $\mathcal{C}_j^i$. This means that this procedure will construct all possible supports in

$$\mathcal{S}_d = \{\text{top}_k(c_1 \cdot \mathbf{v}_1 + \ldots + c_d \cdot \mathbf{v}_d) : c_1, \ldots, c_d \in \mathbb{R}\}$$

B.1.3 Constructing the set $\mathcal{S}_d$

To visit all possible cells $\mathcal{C}_j^i$, we now have to check the intersection points, which are obtained by solving the system of $d-1$ equations

$$\begin{aligned} &[\mathbf{v}(\varphi)]_{i_1} = [\mathbf{v}(\varphi)]_{i_2} = \ldots = [\mathbf{v}(\varphi)]_{i_d} \\ \Leftrightarrow &[\mathbf{v}(\varphi)]_{i_1} = [\mathbf{v}(\varphi)]_{i_2}, \ldots, [\mathbf{v}(\varphi)]_{i_1} = [\mathbf{v}(\varphi)]_{i_d}. \end{aligned} \tag{B.4}$$

We can rewrite the above as

$$\begin{bmatrix} \mathbf{e}_{i_1}^T - \mathbf{e}_{i_2}^T \\ \vdots \\ \mathbf{e}_{i_1}^T - \mathbf{e}_{i_d}^T \end{bmatrix} \mathbf{V}_d \mathbf{c} = \mathbf{0}_{(d-1)\times 1} \tag{B.5}$$

where the solution is the nullspace of the matrix, which has dimension 1.

To explore all possible candidate vectors, we need to visit all cells. To do so, we compute all possible $\binom{n}{d}$ solution intersection vectors $\mathbf{c}_{i_1,\ldots,i_d}$. On each intersection vector we need to compute the locally optimal support set

$$\text{top}_k\left(\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}\right).$$

Then observe that the coordinates $i_1, \ldots, i_d$ of $\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}$ have the same value, since they all satisfy equation (B.5). Let us assume that t of them appear in the set $\text{top}_k\left(\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}\right)$. The, finding the top k supports of all neighboring cell is equivalent to checking all different supports that can be generated by taking all $\binom{d}{t}$ possible t-subsets of the $i_1, \ldots, i_d$ coordinates with respect to $\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}$, while keeping the rest of the elements in $\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}$ in their original ranking, as computed in $\text{top}_k\left(\mathbf{V}_d \mathbf{c}_{i_1,\ldots,i_d}\right)$. This, induces at most $O(\binom{d}{d/2})$ local sortings, i.e., top k supports. All these sortings will eventually be the elements of the $\mathcal{S}_d$ set. The number of all candidate support sets will now be $O(\binom{d}{d/2}\binom{n-1}{d}) = O(n^d)$ and the total computation complexity is $O\left(n^{d+1}\right)$, since for each point we compute the top-k support in linear time $O(n)$.

For completeness the algorithm of the spannogram framework that generates $\mathcal{S}_d$ is given below.

Algorithm 8 Spannogram Algorithm for $\mathcal{S}_d$.

$\mathcal{S}_d = \emptyset$
for all $(i_1, \ldots, i_d) \in \{1, \ldots, n\}^d$ and $s \in \{-1, 1\}$ **do**
  $\mathbf{c} = s \cdot \text{nullspace}\left(\begin{bmatrix} [(\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_2,:} \\ \vdots \\ [\mathbf{V}_d]_{i_1,:} - [\mathbf{V}_d]_{i_d,:} \end{bmatrix}\right)$
  $\mathbf{v} = \mathbf{V}_d^T \mathbf{c}$
  $\mathcal{S} = \text{top}_k(\mathbf{v})$
  $\mathcal{T} = \mathcal{S} - \{i_1, \ldots, i_d\}$
  for all $\binom{d}{k-|\mathcal{T}|}$ subsets $\mathcal{J}$ of $(i_1, \ldots, i_d)$ **do**
    $\mathcal{S}_d = \mathcal{S}_d \bigcup (\mathcal{T} \cup \mathcal{J})$
  end for
end for
Output: $\mathcal{S}_d$.

B.2 Proof of Lemma 1: Going from DkS to DBkS and back

In this subsection we show how a ρ-approximation algorithm for DBkS for arbitrary matrices, implies a 2ρ-approximation for DkS. Our proof goes through a randomized sampling argument.

Algorithm 9 `randombipartite`($\mathcal{G}$)

$\mathcal{L} = \emptyset$, $\mathcal{R} = \emptyset$
draw n fair coins, and assign each of them to the n vertice of the graph.
$\mathcal{L} =$ the set of vertices that corresponds to heads
$\mathcal{R} = \{1, \ldots, n\} \backslash \mathcal{L}$
$\mathcal{G}_{\mathsf{B}} = \mathcal{G}$
delete all edges in $\mathcal{G}_{\mathsf{B}}(\mathcal{L})$ and $\mathcal{G}_{\mathsf{B}}(\mathcal{R})$
Output: $\mathcal{G}_{\mathsf{B}}$

B.2.1 Proof of Lemma 1: Randomized Reduction

Let us denote by $\mathcal{G}(\mathcal{S})$ the subgraph in $\mathcal{G}$ induced by a vertex set $\mathcal{S}$. Let the adjacency matrix of the bipartite graph created by `randombipartite`($\mathcal{G}$) be $\mathcal{G}_{\mathsf{B}}$

$$\mathbf{A}_{\mathsf{B}} = \begin{bmatrix} \mathbf{0}_{n_1 \times n_2} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0}_{n_2 \times n_1} \end{bmatrix},$$

where $n_1 + n_2 = n$. In the following, we refer to $\mathbf{B}$ as the bi-adjacency matrix of the bipartite graph $\mathcal{G}_\mathsf{B}$. Moreover, we denote as $\mathcal{L}$ and $\mathcal{R}$ the two disjoint vertex sets of a bipartite graph.

Before we proceed let us state a simple property on the quadratic form of bipartite graphs.

Proposition 2. *Let* $\mathbf{A}_\mathsf{B} = \begin{bmatrix} \mathbf{0}_{n_1 \times n_2} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0}_{n_2 \times n_1} \end{bmatrix}$ *be the adjacency matrix of a bipartite graph. Then, for any subset of vertices* $\mathcal{S}$*, we have that* $\mathcal{S} = \mathcal{S}_l \cup \mathcal{S}_r$*, with* $\mathcal{S}_l = \mathcal{S} \cap \mathcal{L}$ *and* $\mathcal{S}_r = \mathcal{S} \cap \mathcal{R}$*. Moreover,*

$$\mathbf{1}_\mathcal{S}^T \mathbf{A}_\mathsf{B} \mathbf{1}_\mathcal{S} = 2 \cdot \mathbf{1}_{\mathcal{S}_l}^T \mathbf{B} \mathbf{1}_{\mathcal{S}_r}.$$

Proof. It is easy to see that $\mathcal{S}_l$ and $\mathcal{S}_r$ are the vertex subsets of $\mathcal{S}$ that correspond to either the left or right nodes of the bipartite graph. Since the two sets are disjoint, we have

$$\mathbf{1}_\mathcal{S} = \mathbf{1}_{\mathcal{S}_l} + \mathbf{1}_{\mathcal{S}_r}.$$

Then, the quadratic forms on $\mathbf{A}_\mathsf{B}$ can be equivalently rewritten as bilinear forms on $\mathbf{B}$:

$$\mathbf{1}_\mathcal{S}^T \mathbf{A}_\mathsf{B} \mathbf{1}_\mathcal{S} = (\mathbf{1}_{\mathcal{S}_l} + \mathbf{1}_{\mathcal{S}_r})^T \begin{bmatrix} \mathbf{0}_{n_1 \times n_2} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0}_{n_2 \times n_1} \end{bmatrix} (\mathbf{1}_{\mathcal{S}_l} + \mathbf{1}_{\mathcal{S}_r}) = \mathbf{1}_{\mathcal{S}_r} \mathbf{B}^T \mathbf{1}_{\mathcal{S}_l} + \mathbf{1}_{\mathcal{S}_l} \mathbf{B} \mathbf{1}_{\mathcal{S}_r} = 2 \cdot \mathbf{1}_{\mathcal{S}_l}^T \mathbf{B} \mathbf{1}_{\mathcal{S}_r}.$$

□

Due to the above, we consider the following bilinear optimization problem

$$\{\mathcal{X}_\mathsf{B}, \mathcal{Y}_\mathsf{B}\} = \arg \max_{\substack{|\mathcal{X}|=k_1 \\ |\mathcal{Y}|=k_2 \\ k_1+k_2=k}} \mathbf{1}_\mathcal{X}^T \mathbf{B} \mathbf{1}_\mathcal{Y}, \tag{B.6}$$

where the constraint $k_1 + k_2 = k$ forces the left and right vertices to induce a k-subgraph. Due to Proposition 2, the two vertex sets are disjoint, i.e., $\mathcal{X}_\mathsf{B} \cap \mathcal{Y}_\mathsf{B} = \emptyset$, since the columns and rows of $\mathbf{B}$ index two disjoint vertex sets $\mathcal{L}$ and $\mathcal{R}$, respectively.

Let $\mathcal{S}_\mathsf{B} = \mathcal{X}_\mathsf{B} \cup \mathcal{Y}_\mathsf{B}$ be the k vertices in the union of $\mathcal{X}_\mathsf{B}$ and $\mathcal{Y}_\mathsf{B}$. Then, we will relate the density of $\mathcal{S}_\mathsf{B}$ on the original graph $\mathcal{G}$ to the bipartite we obtain from `randombipartite`($\mathcal{G}$).

Proposition 3. *The density of $\mathcal{S}_{\mathsf{B}}$, the densest k-subgraph of $\mathcal{G}_{\mathsf{B}}$, on the original graph $\mathcal{G}$, is at least*

$$\mathsf{den}(\mathcal{S}_{\mathsf{B}}) = \frac{\mathbf{1}_{\mathcal{S}_{\mathsf{B}}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}_{\mathsf{B}}}}{k} \geq \frac{\mathbf{1}_{\mathcal{S}_{\mathsf{B}}}^T \mathbf{A}_{\mathsf{B}} \mathbf{1}_{\mathcal{S}_{\mathsf{B}}}}{k} = 2 \cdot \frac{\mathbf{1}_{\mathcal{X}_{\mathsf{B}}}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}_{\mathsf{B}}}}{k}.$$

Proof. The result follows immediately by the nonnegativity of the entries in $\mathbf{A}$, and the fact that $\mathbf{A}_{\mathsf{B}}$ contains a subset of the entries of $\mathbf{A}$. The last equality follows from Proposition 2. □

We will now show that $\mathsf{den}(\mathcal{S}_{\mathsf{B}})$ is at least $\mathsf{opt}/2$, in expectation. We will use this fact to show that, if we solve DBkS on $\frac{\log n}{\delta}$ graphs independently created using `randombipartite`($\mathcal{G}$), and by keeping the best solution among them, then the extracted k-subgraph has with high probability density $\mathsf{opt}/(2+\delta)$.

Proposition 4. *Let $\mathcal{G}_{\mathsf{B}}$ be the output of `randombipartite`($\mathcal{G}$). Then, there exists in $\mathcal{G}_{\mathsf{B}}$, a k-subgraph that contains $\frac{k \cdot \mathsf{opt}}{2}$ edges, in expectation.*

Proof. First observe that we can represent the edges of $\mathcal{G}_{\mathsf{B}}$ as random variables $X_{i,j}$. If (i,j) is not an edge in $\mathcal{G}$, then $X_{i,j}$ will be 0 with probability 1. If however (i,j) is an edge in $\mathcal{G}$, then $X_{i,j}$ is 1, i.e., appears in $\mathcal{G}_{\mathsf{B}}$, with the same probability that one of its vertices lands in $\mathcal{L}$, while the second is in $\mathcal{R}$. It is easy to find that this probability is $\Pr\{X_{i,j} = 1\} = 1/2$. Hence,

$$X_{i,j} = \begin{cases} 0, & \text{if } (i,j) \text{ not an edge in } \mathcal{G}, \\ Z, & \text{if } (i,j) \text{is an edge in } \mathcal{G}, \end{cases} \tag{B.7}$$

where Z is a Bernoulli$(1/2)$ random variable.

Now let $\mathcal{S}_*$ denote the vertex set of the densest k-subgraph on the original graph $\mathcal{G}$, that has density $\mathsf{den}(\mathcal{S}_*) = \mathsf{opt}$. Observe that for that subgraph we have

$$\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{S}_*} = \sum_{i,j \in \mathcal{S}_*} A_{i,j} = k \cdot \mathsf{opt}.$$

Let $\mathbf{A}_{\mathsf{B}}$ denote the adjacency matrix of the bipartite graph $\mathcal{G}_{\mathsf{B}}$. Then, we have that the expected quadratic form on the new adjacency $\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A}_{\mathsf{B}} \mathbf{1}_{\mathcal{S}_*}$ is:

$$E\left\{\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A}_{\mathsf{B}} \mathbf{1}_{\mathcal{S}_*}\right\} = E\left\{\sum_{i,j \in \mathcal{S}_*} X_{i,j}\right\} = E\left\{\sum_{\substack{i,j \in \mathcal{S}_* \\ (i,j) \in \mathcal{G}}} Z\right\} = \frac{1}{2} \cdot \sum_{i,j \in \mathcal{S}_*} A_{i,j} = \frac{k \cdot \mathsf{opt}}{2}.$$

□

We will now show that if we run `randombipartite`($\mathcal{G}$) a total number of $3\log n \cdot \log\log n$ times, then with high probability, at least one $\mathcal{G}_{\mathsf{B}}$ will contain a k-subgraph with density at least $0.5 \cdot \mathsf{opt}$. This will imply that the densest k subgraph of $\mathcal{G}_{\mathsf{B}}$ will have density *at least* $0.5 \cdot \mathsf{opt}$.

Algorithm 10 `DkS_2_approx`($\mathcal{G}$, δ)

for $i = 1 : \frac{\log n}{\delta}$ **do**
  $\mathcal{G}_{\mathsf{B}^i} = \texttt{randombipartite}(\mathcal{G})$
  $\mathbf{B}^i =$ biadjacency of $\mathcal{G}_{\mathsf{B}^i}$
  $\{\mathcal{X}^i, \mathcal{Y}^i\} = \arg\max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2, k_1+k_2=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{B}^i \mathbf{1}_{\mathcal{Y}}$
end for
$\{\mathcal{X}_{\mathsf{B}}, \mathcal{Y}_{\mathsf{B}}\} = \arg\max_i \mathbf{1}_{\mathcal{X}^i \cup \mathcal{Y}^i}^T \mathbf{A} \mathbf{1}_{\mathcal{X}^i \cup \mathcal{Y}^i}$
Output: $\mathcal{S}_{\mathsf{B}} = \mathcal{X}_{\mathsf{B}} \cup \mathcal{Y}_{\mathsf{B}}$

Proposition 5. *Then, with probability at least* $1 - \frac{1}{n}$*, we have*

$$\mathbf{1}_{\mathcal{S}_{\mathsf{B}}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}_{\mathsf{B}}} \geq \frac{1-\delta}{2} \cdot \mathsf{opt}.$$

Proof. In this proof we will use the reverse Markov Inequality which states that for any random variable X, such that $X \leq m$, then, for any $a \leq E\{X\}$, we have

$$\Pr\{X \leq a\} \leq \frac{m - E\{X\}}{m - a}.$$

Let $\mathcal{S}_*$ denote the densest k-subgraph for $\mathcal{G}$. Here, our random variable will be the the quadratic form

$$X = \mathbf{1}_{\mathcal{S}_*}^T \mathbf{A}_{\mathsf{B}}^i \mathbf{1}_{\mathcal{S}_*}.$$

Due to Proposition 4, we have that $E\{X\} = 0.5 \cdot k \cdot \mathsf{opt}$. Hence, set $m = k \cdot \mathsf{opt}$ and $\alpha = \frac{k \cdot \mathsf{opt}}{2} - \delta \cdot \frac{k \cdot \mathsf{opt}}{2}$ to obtain:

$$\Pr\left\{X \leq \frac{k \cdot \mathsf{opt}}{2} - \delta \cdot \frac{k \cdot \mathsf{opt}}{2}\right\} \leq \frac{k \cdot \mathsf{opt} - k \cdot \mathsf{opt}/2}{k \cdot \mathsf{opt} - \frac{k \cdot \mathsf{opt}}{2} + \delta \cdot \frac{k \cdot \mathsf{opt}}{2}} = \frac{1}{1+\delta}$$

Now, observe that if we want to have with probability $1 - \frac{1}{n}$ at least one graph $\mathcal{G}_{B^i}$ where

$$\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A}_{\mathsf{B}}^i \mathbf{1}_{\mathcal{S}_*} > \frac{1-\delta}{2} \cdot k \cdot \mathsf{opt},$$

we need to draw l graphs, such that

$$\left(\frac{1}{1+\delta}\right)^l \leq \frac{1}{n}$$

which yields $l = \frac{\log n}{\log(1+\delta)}$. Since $\delta \in (0,1)$, we have that a number of

$$\frac{\log n}{\delta}$$

draws suffices (assuming the base-2 logarithm), so that

$$\max_i \mathbf{1}^T_{\mathcal{X}^i \cup \mathcal{Y}^i} \mathbf{A}^i_{\mathsf{B}} \mathbf{1}_{\mathcal{X}^i \cup \mathcal{Y}^i} \geq \max_i \mathbf{1}^T_{\mathcal{S}_*} \mathbf{A}^i_{\mathsf{B}} \mathbf{1}_{\mathcal{S}_*} \geq \frac{1-\delta}{2} \mathbf{1}^T_{\mathcal{S}_*} \mathbf{A} \mathbf{1}_{\mathcal{S}_*}.$$

□

Proposition 5 establishes Lemma 2. What we show in our approximation results is that we can compute a solution with density at least

$$\frac{1-\delta}{2} \cdot \mathsf{opt} - 2|\lambda_{d+1}|,$$

where λ_{d+1} is the $d+1$ absolutely largest eigenvalue of the adjacency matrix $\mathbf{A}$.

B.3 Proof of Theorem 1

B.3.1 Low-rank DBkS on bipartite graphs and rectangular matrices

The first important technical proposition that we show, is that we can solve DBkS for any constant rank *rectangular* matrix $\mathbf{B}$ of dimensions $n_1 \times n_2$.

Proposition 6. *Let* $\mathbf{B}$ *be any matrix of size* $n_1 \times n_2$ *and let*

$$\mathbf{B}_d = \sum_{i=1}^{d} \sigma_i \mathbf{v}_i \mathbf{u}_i^T$$

be its singular value decomposition, where $\mathbf{v}_i$ *and* $\mathbf{u}_i$ *is the left and right singular vectors corresponding to the* i*th largest singular value* $\sigma_i(\mathbf{B})$*. Then, we can solve the following problem*

$$\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2} \mathbf{1}^T_{\mathcal{X}} \mathbf{B}_d \mathbf{1}_{\mathcal{Y}}.$$

in time $O(\min\{n_1, n_2\}^{d+1})$.

Proof. For simplicity we assume that the left singular vectors are scaled by their singular values, hence

$$\mathbf{B}_d = \mathbf{v}_1\mathbf{u}_1^T + \ldots + \mathbf{v}_d\mathbf{u}_d^T.$$

Let us without loss of generality assume that $n_1 \leq n_2$.

We wish to solve:

$$\max_{|\mathcal{X}|=k_1,|\mathcal{Y}|=k_2} \mathbf{1}_{\mathcal{X}}^T \left(\mathbf{v}_1\mathbf{u}_1^T + \ldots + \mathbf{v}_d\mathbf{u}_d^T\right)\mathbf{1}_{\mathcal{Y}}. \tag{B.8}$$

Observe that we can rewrite (B.8) in the following way

$$\max_{|\mathcal{X}|=k_1,|\mathcal{Y}|=k_2} \mathbf{1}_{\mathcal{X}}^T \left[\mathbf{v}_1 \cdot \underbrace{(\mathbf{u}_1^T\mathbf{1}_{\mathcal{Y}})}_{c_1} + \ldots + \mathbf{v}_d \cdot \underbrace{(\mathbf{u}_d^T\mathbf{1}_{\mathcal{Y}})}_{c_d}\right] = \max_{|\mathcal{Y}|=k_2} \left(\max_{|\mathcal{X}|=k_1} \mathbf{1}_{\mathcal{X}}^T\mathbf{v}_{\mathcal{Y}}\right),$$

where $\mathbf{v}_{\mathcal{Y}} = \mathbf{v}_1 \cdot c_1 + \ldots + \mathbf{v}_d \cdot c_d$ is an n_1-dimensional vector generated by the d-dimensional subspace spanned by $\mathbf{v}_1, \ldots, \mathbf{v}_d$.

We will now make a key observation: for every fixed vector $\mathbf{v}_{\mathcal{Y}}$, the index set $\mathcal{X}$ that maximizes $\mathbf{1}_{\mathcal{X}}^T\mathbf{v}_{\mathcal{Y}}$ can be easily computed. It is not hard to see that for any fixed vector $\mathbf{v}_{\mathcal{Y}}$, the k_1-subset $\mathcal{X}$ that maximizes

$$\mathbf{1}_{\mathcal{X}}^T\mathbf{v}_{\mathcal{Y}} = \sum_{i\in\mathcal{X}} [\mathbf{v}_{\mathcal{Y}}]_i$$

corresponds to either the set of k_1 largest or k_1 smallest signed coordinates of $\mathbf{v}_{\mathcal{Y}}$. That is, the locally optimal sets are either $\text{top}_{k_1}(\mathbf{v}_{\mathcal{Y}})$ or $\text{top}_{k_1}(-\mathbf{v}_{\mathcal{Y}})$.

We now wish to find all possible locally optimal sets $\mathcal{X}$. If we could possibly check all vectors $\mathbf{v}_{\mathcal{Y}}$, then we could find all locally optimal index sets $\text{top}_{k_1}(\pm\mathbf{v}_{\mathcal{Y}})$.

Let us denote as $\mathcal{S}_d$ the set of all k_1-sized sets $\mathcal{X}$ that are the optimal solutions of the inner maximization of in the above, for *any* vector $\mathbf{v}$ in the span of $\mathbf{v}_1, \ldots, \mathbf{v}_d$

$$\mathcal{S}_d = \{\text{top}_{k_1}(\pm[\mathbf{v}_1 \cdot c_1 + \ldots + \mathbf{v}_d \cdot c_d]) : c_1, \ldots, c_d \in \mathbb{R}\}.$$

Clearly, this set contains all possible locally optimal $\mathcal{X}$ sets of the form $\text{top}_{k_1}(\mathbf{v}_{\mathcal{Y}})$. Therefore, we can rewrite DBkS on $\mathbf{B}_d$ as

$$\max_{|\mathcal{Y}|=k_2} \max_{\mathcal{X}\in\mathcal{S}_d} \mathbf{1}_{\mathcal{X}}^T\mathbf{B}_d\mathbf{1}_{\mathcal{Y}}. \tag{B.9}$$

The above problem can now be solved in the following way: for every set $\mathcal{X} \in \mathcal{S}_d$ find the locally optimal set $\mathcal{Y}$ that maximizes $\mathbf{1}_{\mathcal{X}}^T\mathbf{B}_d\mathbf{1}_{\mathcal{Y}}$. Again, this will either be

$\text{top}_{k_2}(-\mathbf{B}_d\mathbf{1}_{\mathcal{X}})$ or $\text{top}_{k_2}(\mathbf{B}_d\mathbf{1}_{\mathcal{X}})$. Then, we simply need to test all such $\mathcal{X}, \mathcal{Y}$ pairs on $\mathbf{B}_d$ and keep the optimizer.

Due to the above, the problem of solving DBkS on the rectangular matrix $\mathbf{B}_d$ is equivalent to constructing the set of k_1-supports $\mathcal{S}_d$, and then finding the optimal solution in that set. How large can $\mathcal{S}_d$ be and can we construct it in polynomial time? As we showed in the first section of this appendix, this problem is equivalent to that solved in [10], hence the set has size $O(\binom{n_1}{d})$ and can be constructed in time $O(n_1^{d+1})$.

Observe that in the above we could have equivalently solved the problem by finding all the top k_2 sets in the span of $\mathbf{u}_1, \ldots, \mathbf{u}_d$, say that they belong in set $\mathcal{S}'_d$. Then, we could solve the problem by finding for each k_2 sized set $\mathcal{Y} \in \mathcal{S}'_d$ the optimal k_1 sized set $\mathcal{X}$. Both approaches are the same, and the one with the smallest dimension is selected to reduce the computational complexity.

The algorithm that solves the problem for rectangular matrices is given below.

Algorithm 11 low-rank approximations for DBkS

`lowrankDBkS(`k_1`, `k_2`, `d`, `$\mathbf{B}$`)`
$[\mathbf{V}_d, \Sigma_d, \mathbf{U}_d] = \texttt{SVD}(\mathbf{B}, d)$
$\mathcal{S}_d = \texttt{Spannogram}(k_1, \mathbf{V}_d)$
$\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg\max_{|\mathcal{Y}|=k_2} \max_{\mathcal{X}\in\mathcal{S}_d} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \Sigma_d \mathbf{U}_d^T \mathbf{1}_{\mathcal{Y}}$
Output: $\{\mathcal{X}_d, \mathcal{Y}_d\}$

`Spannogram(`k_1`, `$\mathbf{V}_d$`)`
$\mathcal{S}_d = \{\text{top}_k(\mathbf{v}) : \mathbf{v} \in \text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_d)\}$
Output: $\mathcal{S}_d$.

□

B.3.2 Bipartite graphs part of Theorem 1

In our following derivations, for both cases of a rectangular and square symmetric matrices, we consider the same notation of the output solution and out-

put density for simplicity:

$$\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{|\mathcal{X}|=k, |\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}} \text{ and } \mathsf{opt}_d^{\mathsf{B}} = \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_d}}{k},$$

$$\{\mathcal{X}_d, \mathcal{Y}_d\} = \arg \max_{k_1,k_2:k_1+k_2=k} \max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2} \mathbf{1}_{\mathcal{X}}^T \mathbf{B}_d \mathbf{1}_{\mathcal{Y}} \text{ and } \mathsf{opt}_d^{\mathsf{B}} = 2\frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}_d}}{k}.$$

Moreover, as a reminder the optimal solutions and densities for the problems of interest (DkS, DBkS on $\mathbf{A}$, and DBkS on $\mathbf{B}$) are

$$\mathcal{S}* = \arg \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A} \mathbf{1}_{\mathcal{S}} \text{ and } \mathsf{opt} = \frac{\mathbf{1}_{\mathcal{S}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{S}_*}}{k},$$

$$\{\mathcal{X}_*, \mathcal{Y}_*\} = \arg \max_{|\mathcal{X}|=k, |\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}} \text{ and } \mathsf{opt}^{\mathsf{B}} = \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_*}}{k},$$

$$\{\mathcal{X}_*, \mathcal{Y}_*\} = \arg \max_{k_1,k_2:k_1+k_2=k} \max_{|\mathcal{X}|=k_1, |\mathcal{Y}|=k_2} \mathbf{1}_{\mathcal{X}}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}} \text{ and } \mathsf{opt}^{\mathsf{B}} = 2\frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}_*}}{k}.$$

We continue with bounding the distance between the optimal solution for DBkS and rank-d optimal solution pair $\{\mathcal{X}_d, \mathcal{Y}_d\}$. We have the following result, which is essentially Lemma 2 of our main paper.

Proposition 7. *For any matrix* $\mathbf{A}$*, we have*

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot |\lambda_{d+1}|. \tag{B.10}$$

Moreover, for any rectangular matrix $\mathbf{B}$*, we have*

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot \sigma_{d+1}. \tag{B.11}$$

Proof. Let $\mathcal{X}_*, \mathcal{Y}_*$ be the optimal solution of DBkS on $\mathbf{A}$ and let $\mathcal{X}_d, \mathcal{Y}_d$ be the optimal solution of DBkS on the rank-d matrix $\mathbf{A}_d$. Then, we have

$$\begin{aligned} \mathsf{opt}_d^{\mathsf{B}} = \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_d}}{k} = \frac{\mathbf{1}_{\mathcal{X}_d}^T (\mathbf{A}_d + \mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_d}}{k} = \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\mathbf{1}_{\mathcal{X}_d}^T (\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_d}}{k} \\ \geq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} - \frac{\|\mathbf{1}_{\mathcal{X}_d}\|_2 \cdot \|(\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_d}\|_2}{k} \geq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} - |\lambda_{d+1}|, \end{aligned} \tag{B.12}$$

where the first inequality comes due to Cauchy-Schwarz and the second due to the fact that the norm of the indicator vector is k and the operator norm of $\mathbf{A} - \mathbf{A}_d$ is equal to the $d+1$ largest eigenvalue of $\mathbf{A}$.

Moreover, we have that

$$\begin{aligned}\mathsf{opt}^{\mathsf{B}} &= \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A}\mathbf{1}_{\mathcal{Y}_*}}{k} = \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A}_d + \mathbf{A} - \mathbf{A}_d)\mathbf{1}_{\mathcal{Y}_*}}{k} = \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_*}}{k} + \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A} - \mathbf{A}_d)\mathbf{1}_{\mathcal{Y}_*}}{k} \\ &\leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A} - \mathbf{A}_d)\mathbf{1}_{\mathcal{Y}_*}}{k} \\ &\leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\|\mathbf{1}_{\mathcal{X}_*}\|_2 \, \|(\mathbf{A} - \mathbf{A}_d)\mathbf{1}_{\mathcal{Y}_*}\|_2}{k} \leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + |\lambda_{d+1}| \end{aligned} \tag{B.13}$$

where the first inequality comes due to the fact that $\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d} \geq \mathbf{1}_{\mathcal{X}_*}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_*}$ and the second and third are similar to the previous bound. We can now combine the above two bound to obtain:

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot |\lambda_{d+1}|. \tag{B.14}$$

In the exact same way, we can obtain the result for rectangular matrices. □

The above proposition, combined with Proposition 2 give us the bipartite part of Theorem 1, where $\mathsf{opt}^{\mathsf{B}} = \mathsf{opt}$, that is

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2 \cdot |\lambda_{d+1}| = \mathsf{opt} - 2 \cdot |\lambda_{d+1}|.$$

B.3.3 Graphs with their first d eigenvalues positive part of Theorem 1

To establish the part about graphs with the d largest eigenvalues being positive, we use the following result.

Proposition 8. *If $\mathbf{A}_d$ is positive semidefinite, then*

$$\max_{|\mathcal{X}|=k} \frac{\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{X}}}{k} = \max_{|\mathcal{X}|=k} \max_{|\mathcal{Y}|=k} \frac{\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}}{k}$$

Proof. This is easy to see by the fact that for any two sets $\mathcal{X}, \mathcal{Y}$ we have

$$\begin{aligned}\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}} &\leq \max_{|\mathcal{X}|=k,|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}} = \max_{|\mathcal{X}|=k,|\mathcal{Y}|=k} \mathbf{1}_{\mathcal{X}}^T \mathbf{V}_d \Lambda_d^{1/2} \Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{Y}} \\ &\leq \max_{|\mathcal{X}|=k,|\mathcal{Y}|=k} \max \left\{ \|\Lambda_d^{1/2} \mathbf{V}_d \mathbf{1}_{\mathcal{X}}\|^2, \;\; \|\Lambda_d^{1/2} \mathbf{V}_d \mathbf{1}_{\mathcal{Y}}\| \right\} \\ &\leq \max_{|\mathcal{X}|=k,|\mathcal{Y}|=k} \max \left\{ \mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{X}}, \;\; \mathbf{1}_{\mathcal{Y}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}} \right\} \\ &\leq \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}}\end{aligned}$$

where the second inequality comes due to the Cauchy-Schwarz inequality. □

We can combine the above proposition with the first part of Proposition 7 to obtain that

$$\mathsf{opt}_d = \mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt} - 2|\lambda_{d+1}(\mathbf{A})|$$

when $\mathbf{A}_d$ is positive semidefinite.

B.3.4 Arbitrary graphs part of Theorem 1

In the next proposition, we show how to translate a low-rank approximation of $\mathbf{A}$ after we used the random sampling of `randombipartite`($\mathcal{G}$). We need this result to to establish the general result of Theorem 1, by connecting the previous spectral bound, with the 2 loss in approximation between DBkS and DkS.

Proposition 9. *Let $\mathbf{A}$ be the adjacency matrix of a graph. Moreover, let the matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ be such that $\mathbf{B} = \mathbf{P}_1\mathbf{A}\mathbf{P}_2$ is the bi-adjecency created by each loop of `randombipartite`($\mathcal{G}$), where $\mathbf{P}_1$ is an $n_1 \times n$ matrix indexing the left vertices of the graph, and $\mathbf{P}_2$ is an $n \times n_2$ sampling matrix that indexes the right vertices of the sub-sampled graph. Then,*

$$\mathsf{opt}_d^{\mathsf{B}} \geq \mathsf{opt}^{\mathsf{B}} - 2|\lambda_{d+1}(\mathbf{A})|,$$

where $\mathsf{opt}^{\mathsf{B}}$ is the maximum density on $\mathbf{B} = \mathbf{P}_1\mathbf{A}\mathbf{P}_2$.

Proof. Let without loss of generality assume that $\mathbf{B}$ will be the bipartite subgraph between the first n_1 and the remaining $n_2 = n - n_1$ vertices, such that

$$\mathbf{A} = \begin{bmatrix} \mathbf{C} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{D} \end{bmatrix} \tag{B.15}$$

Then there are two sampling matrices that pick the corresponding columns and rows

$$\mathbf{P}_1 = [\mathbf{I}_{n_1 \times n_1} \quad \mathbf{0}_{n_1 \times n_2}] \text{ and } \mathbf{P}_2 = \begin{bmatrix} \mathbf{0}_{n_1 \times n_2} \\ \mathbf{I}_{n_2 \times n_2} \end{bmatrix}$$

Then, instead of working on the matrix that is the rank-d best fit for $\mathbf{B}$, we work on

$$\mathbf{B}_d = \mathbf{P}_1\mathbf{A}_d\mathbf{P}_2.$$

Now, we use the bounding techniques of our previous derivations:

$$\begin{aligned}
\mathsf{opt}_d &= \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{B} \mathbf{1}_{\mathcal{Y}_d}}{k} = \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{P}_1 (\mathbf{A}_d + \mathbf{A} - \mathbf{A}_d) \mathbf{P}_2 \mathbf{1}_{\mathcal{Y}_d}}{k} \\
&= \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{P}_1 \mathbf{A}_d \mathbf{P}_2 \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{P}_1 (\mathbf{A} - \mathbf{A}_d) \mathbf{P}_2 \mathbf{1}_{\mathcal{Y}_d}}{k} \\
&\geq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{P}_1 \mathbf{A}_d \mathbf{P}_2 \mathbf{1}_{\mathcal{Y}_d}}{k} - \frac{\|\mathbf{1}_{\mathcal{X}_d}\|_2 \cdot \|\mathbf{P}_1 (\mathbf{A} - \mathbf{A}_d) \mathbf{P}_2 \mathbf{1}_{\mathcal{Y}_d}\|_2}{k} \\
&\geq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{P}_1 \mathbf{A}_d \mathbf{P}_2 \mathbf{1}_{\mathcal{Y}_d}}{k} - |\lambda_{d+1}|, \qquad \text{(B.16)}
\end{aligned}$$

where the last step comes due to the fact that $\mathbf{P}_1, \mathbf{P}_2$ their singular values are 1. We can use a similar bound to obtain

$$\mathsf{opt}^{\mathsf{B}} \leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{P}_1 \mathbf{A}_d \mathbf{P}_2 \mathbf{1}_{\mathcal{Y}_d}}{k} + |\lambda_{d+1}|,$$

where $\mathsf{opt}^{\mathsf{B}}$ is the densest k-subgraph on the graph with bi-adjacency matrix $\mathbf{P}_1 \mathbf{A} \mathbf{P}_2$. and combine the above to establish the result. □

We can now use our random sampling Proposition 5 and combine that with Propositions 9, and 7, to establish Theorem 1 for arbitrary graphs.

B.4 Proof of Theorem 2: graphs with highly dense k-subgraphs

We now establish the following a priori spectral bound that holds for any graph.

Lemma 17. *For any unweighted graph $\mathcal{G}$, we have that*

$$|\lambda_d| \leq \sqrt{\frac{2 \cdot m}{d}} \qquad \text{(B.17)}$$

where m is the number of edges in $\mathcal{G}$.

Proof. Observe that

$$d \cdot \lambda_d^2 \leq \sum_{i=1}^{d} \lambda_i^2 \leq \sum_{i=1}^{n} \lambda_i^2 = \|\mathbf{A}\|_F^2 = \sum_{i,j} A_{i,j}^2 = \sum_{i,j} A_{i,j} = 2 \cdot m, .$$

where the second to last equality comes due to the fact that $A_{i,j}^2$ can only be 1 or 0. □

We use this bound and Theorem 1, to obtain a the following result, which is a restatement of Theorem 2.

Proposition 10. *If the densest-k-subgraph contains a constant fraction of all the edges, and $k = \Theta(\sqrt{E})$, then we can approximate DkS within a factor of $2+\epsilon$, in time $n^{O(1/\epsilon^2)}$. If additionally the graph is bipartite, then we can approximate DkS within a factor of $1+\epsilon$.*

Proof. For any arbitrary graph, due to Theorem 1, we have

$$\mathsf{opt}_d \geq \left(\frac{1-\delta}{2}\right) \cdot \mathsf{opt} - 2 \cdot |\lambda_{d+1}|.$$

Since, we assumed that the densest k-subgraph contains a constrant fraction of the edges, this means that $k \cdot \mathsf{opt} = c_1 \cdot m$ for some constant $c > 0$. Moreover, we assumed that $k = c_2 \cdot \sqrt{m}$, for some constant $c_2 > 0$. Hence,

$$c_2 \cdot \sqrt{m} \cdot \mathsf{opt} = c_1 \cdot m \Rightarrow \mathsf{opt} = \frac{c_1}{c_2} \cdot \sqrt{m}.$$

Using Lemma 17, we also get

$$|\lambda_d| \leq \sqrt{\frac{2m}{d}} = \sqrt{\frac{2}{d}} \cdot \frac{c_2}{c_1} \cdot \mathsf{opt}.$$

Combining the above gives us

$$\mathsf{opt}_d \geq \left(\frac{1-\delta}{2}\right) \cdot \mathsf{opt} - 2|\lambda_{d+1}| \geq \left(\frac{1-\delta}{2} - \sqrt{\frac{2}{d}} \cdot \frac{c_2}{c_1}\right) \cdot \mathsf{opt}$$

Hence, if we want $\sqrt{\frac{2}{d}} \cdot \frac{c_2}{c_1} = \frac{\delta}{2}$, we need to set $d = \left\lceil \frac{1}{2} \cdot \frac{c_2^2}{c_1^2} \cdot \frac{1}{\delta^2} \right\rceil = O(\frac{1}{\delta^2})$. Setting $\delta = \frac{\epsilon}{2}$ establishes the result. In a similar way, we obtain the $1+\epsilon$ approximation for bipartite graphs, by using the second bound of Theorem 1.

□

B.5 Proof of Lemma 3: Data Dependent Bounds

Proof. Let $\mathcal{X}_*, \mathcal{Y}_*$ be the optimal solution of DBkS on $\mathbf{A}$ and let Let $\mathcal{X}_d, \mathcal{Y}_d$ be the optimal solution of DBkS on the rank-d matrix $\mathbf{A}_d$. Then, for the first bound we

have

$$\begin{aligned}
\mathsf{opt}^{\mathsf{B}} &= \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}_*}}{k} = \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A}_d + \mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}}{k} \\
&= \frac{\mathbf{1}_{\mathcal{X}_*}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_*}}{k} + \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}}{k} \\
&\leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\mathbf{1}_{\mathcal{X}_*}^T (\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}}{k} \\
&\leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + \frac{\|\mathbf{1}_{\mathcal{X}_*}\|_2 \, \|(\mathbf{A} - \mathbf{A}_d) \mathbf{1}_{\mathcal{Y}_*}\|_2}{k} \\
&\leq \frac{\mathbf{1}_{\mathcal{X}_d}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}_d}}{k} + |\lambda_{d+1}|. \qquad \text{(B.18)}
\end{aligned}$$

The upper bound $k-1$ is trivial by the fact that for any $\mathcal{X}$ and $\mathcal{Y}$ we have

$$\frac{\mathbf{1}_{\mathcal{X}}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}}}{k} \leq \frac{\mathbf{1}_{\mathcal{X}}^T (\mathbf{1}\mathbf{1}^T - \mathbf{I}_n) \mathbf{1}_{\mathcal{Y}}}{k} \leq \frac{k(k-1)}{k} = k - 1.$$

The last bound is simply due to the spectral bound on the bilinear form

$$\frac{\mathbf{1}_{\mathcal{X}}^T \mathbf{A} \mathbf{1}_{\mathcal{Y}}}{k} \leq \frac{\|\mathbf{1}_{\mathcal{X}}\|_2 \cdot \|\mathbf{A}\mathbf{1}_{\mathcal{Y}}\|_2}{k} \leq \lambda_1.$$

□

B.6 Proof of Theorem 3: Nearly-linear Time Algorithm

When the matrix $\mathbf{A}_d$ has mixed signs of eigenvalues, then we have to go through the route of DBkS. However, when $\mathbf{A}_d$ has only positive eigenvalues, then it is easy to show that solving the bilinear problem on $\mathbf{A}_d$ is equivalent to solving

$$\max_{|\mathcal{X}|=k} \frac{\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{X}}}{k}.$$

This is the DkS low-rank problem, that now can be solved by our algorithm. We show that when this spectral scenario holds, we can speed up computations tremendously, by the use of a simple randomization.

Let us first remind the fact that DBkS and DkS are equivalent for positive semidefinite matrices. We will show here how $\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}}$ can be approximately in time nearly-linear in n, by only introducing a small relative approximation error. Our approximation will use ϵ-nets.

Definition 8. *(ϵ-net) Let $\mathbb{S}^d = \{\mathbf{c} \in \mathbb{R}^d : \|\mathbf{c}\|^2 = 1\}$ be the surface of the d-dimensional sphere. An ϵ-net of $\mathbb{S}^d$ is a finite set $\mathcal{N}_\epsilon^d \subset \mathbb{S}^d$ such that*

$$\forall \mathbf{c} \in \mathbb{S}^d \;\; \exists \;\; \widehat{\mathbf{c}} \in \mathcal{N}_\epsilon^d : \|\mathbf{c} - \widehat{\mathbf{c}}\|_2 \leq \epsilon.$$

We now show that we can solve our optimization, via the use of ϵ-nets, which we construct in the next subsection.

Proposition 11. *Let $\mathcal{N}_\epsilon^d$ be an ϵ-net of $\mathbb{S}^d$. Then,*

$$(1-\epsilon)^2 \cdot \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}} \leq \max_{\mathbf{c} \in \mathcal{N}_\epsilon^d} \max_{|\mathcal{S}|=k} \left(\mathbf{c}^T \Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\right)^2 \leq \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}}. \tag{B.19}$$

Proof. Let $\mathbf{c}$ be a $d \times 1$ unit length vector, *i.e.*, $\|\mathbf{c}\|_2 = 1$. Then, by the Cauchy-Schwartz inequality we have

$$(\mathbf{c}^T \Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}})^2 \leq \|\Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\|_2^2.$$

We can get equality in the previous bound for a unit norm $\mathbf{c}$ co-linear to $\Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}$. Therefore, we have

$$\|\Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\|_2^2 = \max_{\|\mathbf{c}\|_2=1} (\mathbf{c}^T \Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}})^2. \tag{B.20}$$

Hence,

$$\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}} = \max_{|\mathcal{S}|=k} \|\Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\|_2^2 = \max_{|\mathcal{S}|=k} \max_{\|\mathbf{c}\|_2=1} (\mathbf{c}^T \Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}})^2. \tag{B.21}$$

We can now obtain the upper bound of the proposition, since $\mathcal{N}_\epsilon^d \subseteq \mathbb{S}^d$. Now for the lower bound, let $(\mathbf{1}_{\mathcal{S}_d}, \mathbf{c}_d)$ denote the optimal solution of the above maximization, such that

$$\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}} = (\mathbf{c}_d {\mathbf{V}_d}^T \mathbf{1}_{\mathcal{S}_d})^2.$$

Then, there exists a vector $\widehat{\mathbf{c}}$ in the ϵ-net $\mathcal{N}_\epsilon^d$, such that $\mathbf{c}_d = \widehat{\mathbf{c}} + \mathbf{r}$, with $\|\mathbf{r}\| \leq \epsilon$. Then,

$$\begin{aligned}\sqrt{\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}}} &= {\mathbf{c}_d}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d} = (\widehat{\mathbf{c}} + \mathbf{r})^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d} \\ &\overset{(\alpha)}{\leq} \widehat{\mathbf{c}}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d} + \epsilon \cdot \|\mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d}\| = \widehat{\mathbf{c}}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d} + \epsilon \cdot \sqrt{\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}}},\end{aligned} \tag{B.22}$$

where (α) is due to the triangle inequality, then the Cauchy-Schwartz inequality, and then the fact that $\|\mathbf{r}\| \leq \epsilon$. From the above inequality we get

$$(1-\epsilon)^2 \max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{S}} \leq \left(\widehat{\mathbf{c}}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}_d}\right)^2 \leq \max_{\mathbf{c}\in\mathcal{N}_\epsilon^d} \max_{|\mathcal{S}|=k} \left(\mathbf{c}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\right)^2$$

which concludes the proof. □

The importance of the above proposition lies in the fact that for a fixed $\mathbf{c}$ we can easily solve the problem

$$\max_{|\mathcal{S}|=k} \left(\mathbf{c}^T \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\right)^2.$$

Observe that the above optimization is the same problem that we had to solve for a fixed $\mathcal{Y}$ in Section 3. This inner product is maximized when $\mathcal{S}$ picks the largest, or smallest k elements of the n-dimensional vector $\mathbf{c}^T \mathbf{V}_d^T$. The complexity to do that is linear $O(n)$ [26].

It is now obvious that the number of elements, and the complexity to construct $\mathcal{N}_\epsilon^d$ is important. In the next subsection, we show how to build such a net, using similar random coding arguments to [105]. Let $\mathcal{N}_\epsilon^d$ be a set of vectors drawn uniformly on the sphere (the cardinality is determined in the next subsection and will be $O(\frac{1}{\epsilon^d} \cdot \log(\frac{1}{\epsilon \cdot \delta}))$. Our algorithm operates as follows: First we draw a set of $|\mathcal{N}_\epsilon^d|$ random vectors, and then we find the corresponding optimal $\mathcal{S}$, by solving

$$\max_{\mathbf{c}\in\mathcal{M}_\epsilon} \max_{|\mathcal{S}|=k} \left(\mathbf{c}^T \Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}}\right)^2.$$

This can be done in time $O(|\mathcal{N}_\epsilon^d| \cdot n)$. Then, among all these solutions, with probability $1-\delta$, the best solution satisfies the bound of the proposition. The randomized algorithm is given below for completeness.

Algorithm 12 Randomized Spannogram

1: `Spannogram_approx(`k, $\mathbf{V}_d, \Lambda_d$`)`
2: $\mathcal{S}_d = \emptyset$
3: **for** $i = 1 : |\mathcal{N}_\epsilon^d|$ **do**
4: $\quad \mathbf{v} = (\Lambda_d^{1/2} \cdot \mathbf{V}_d)^T \cdot \texttt{randn}(d, 1)$
5: $\quad \mathcal{S}_d = \mathcal{S}_d \cup \text{top}_k(\mathbf{v}) \cup \text{top}_k(-\mathbf{v})$
6: **end for**
7: **Output:** $\arg\max_{\mathcal{S}\in\mathcal{S}_d} \left\| \Lambda_d^{1/2} \mathbf{V}_d^T \mathbf{1}_{\mathcal{S}} \right\|$.

Computing $\mathbf{A}_d$ in the first step of the algorithm, can also be done in nearly linear-time in the size of the input $\mathbf{A}$. There is extensive literature on approximating $\mathbf{A}_d$ by a rank-d matrix $\widehat{\mathbf{A}}_d$ such that $\|\mathbf{A}_d - \widehat{\mathbf{A}}_d\|_2 \leq \delta$, in time proportional to the nonzero entries of $\mathbf{A}$ times a logarithmic term, as long as $|\lambda_d/\lambda_{d+1}|$ is at least a constant [43, 45, 86].

B.6.1 A simple ϵ-net construction via random coding principles

We construct an ϵ-net of the d-dimensional sphere by randomly and independently drawing a sufficient number of uniformly distributed points. This construction is essentially studied by Wyner [105] in the asymptotic $d \to \infty$ regime, under a different question: how many random spherical caps are needed to cover a sphere?

The idea behind our construction is simple, and uses to ingredients. First, by a lemma of [101] (p.8, Lemma 5.2) we know that there exist an ϵ-net on the d-dimensional sphere of size at most

$$|\mathcal{N}_\epsilon^d| \leq \left(1 + \frac{2}{\epsilon}\right)^d.$$

Then, we use an elementary balls-and-bin arguments to find the number of vectors that we need to draw at random, so that each point of the net $\mathcal{N}_\epsilon^d$, is ϵ-close to at least one of the vectors that we drew. This set of random vectors will then correspond to a $2 \cdot \epsilon$-net.

More formally, let

$$\mathbb{C}^d(\mathbf{c}_0, \epsilon) = \left\{\mathbf{c} \in \mathbb{S}^d : \|\mathbf{c} - \mathbf{c}_0\| \leq \epsilon\right\},$$

be a spherical cap of $\mathbb{S}^d$ centered at $\mathbf{c}_0$, that includes all vectors within distance ϵ from $\mathbf{c}_0$. Let us now define a set of $m_{\epsilon,d}$ spherical caps for each point $\mathbf{c}_0$ of the set $\mathcal{N}_\epsilon^d$. Then, by the definition of an ϵ-net, the caps centered at the vectors of $\mathcal{N}_\epsilon^d$ cover the sphere:

$$\bigcup_{\mathbf{c}_0 \in \mathcal{N}_\epsilon^d} \mathbb{C}^d(\mathbf{c}_0, \epsilon) = \left\{\mathbf{c} \in \mathbb{S}^d : \|\mathbf{c} - \mathbf{c}_0\| \leq \epsilon\right\} = \mathbb{S}^d.$$

Now, consider an arbitrary point $\widehat{\mathbf{c}_0}$ in some spherical cap $\mathbb{C}^d(\mathbf{c}_0, \epsilon)$. By the triangle inequality, any other point $\mathbf{c}$ in $\mathbb{C}^d(\mathbf{c}_0, \epsilon)$, satisfies

$$\|\mathbf{c} - \widehat{\mathbf{c}_0}\| \leq \|\mathbf{c} - \mathbf{c}_0\| + \|\mathbf{c}_0 - \widehat{\mathbf{c}_0}\| \leq 2\epsilon.$$

The above implies that if we construct a set that contains at least one point from each of the $m_{\epsilon,d}$ caps centered on the vectors of $\mathcal{N}_\epsilon^d$, then this set of points forms a 2ϵ-net. In the following, we do this by randomly drawing a sufficient number of vectors on the sphere.

Let us draw random points uniformly distributed over $\mathbb{S}^d$, by normalizing randomly generated Gaussian vectors distributed according to $N(\mathbf{0}, \mathbf{I}_d)$. A random point falls in one particular cap with probability at least $\frac{1}{m_{\epsilon,d}}$. This is true, since $m_{\epsilon,d}$ spherical caps suffice to cover the surface of the sphere. The probability that some of the caps is empty after we throw m vectors is

$$\Pr\left\{\bigcup_{i=1}^{|\mathcal{N}_\epsilon^d|}\{\text{cap } i \text{ is empty after } m \text{ vector draws}\}\right\} \leq |\mathcal{N}_\epsilon^d| \cdot \left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right)^m. \quad \text{(B.23)}$$

If we wish the probability of this "bad event" to be δ, then we get that the number of m vectors that we need to throw has to satisfy

$$\begin{aligned}
&|\mathcal{N}_\epsilon^d| \cdot \left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right)^m \leq \delta \Rightarrow \log(|\mathcal{N}_\epsilon^d|) + m \cdot \log\left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right) \leq \log(\delta) \\
\Rightarrow &m \geq \frac{\log(\delta) - \log(|\mathcal{N}_\epsilon^d|)}{\log\left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right)} = \frac{\log(|\mathcal{N}_\epsilon^d|/\delta)}{-\log\left(1 - \frac{1}{|\mathcal{N}_\epsilon^d|}\right)} \geq \frac{d\log\left(\frac{1+2\epsilon}{\delta}\right)}{\frac{1}{|\mathcal{N}_\epsilon^d|}} \\
\Rightarrow &m \geq \left(1 + \frac{2}{\epsilon}\right)^d \cdot \left(d \cdot \log\left(1 + \frac{2}{\epsilon}\right) + \log\delta\right)
\end{aligned}$$

Hence, we get the following lemma.

Lemma 18. *Let us draw uniformly at random*

$$\left(1 + \frac{2}{\epsilon}\right)^d \cdot \left(d \cdot \log\left(1 + \frac{2}{\epsilon}\right) + \log\delta\right) = O\left(\frac{1}{\epsilon^d}\log\left(\frac{1}{\epsilon \cdot \delta}\right)\right)$$

vectors on the d-dimensional sphere. Then, with probability at least $1 - \delta$, this set is a $2 \cdot \epsilon$-net of the sphere.

B.7 Vertex Sparsification via Simple Leverage Score Sampling

Our algorithm comes together with a vertex elimination step: after we compute the low-rank approximation matrix $\mathbf{A}_d$, we discard rows and columns of $\mathbf{A}_d$, i.e., vertices, depending on their *weighted leverage scores*. Leverage score sampling has been extensively studied in the literature, for many different applications, where it can provably provide small error bounds, while keeping a small number of features from the original matrix [16, 68].

As we see in the following, this pre-processing step comes with an error guarantee. We show that by throwing away vertices with small leverage scores, can only introduce a provably small error.

Let us define as

$$\ell_i = \left\| \left[\mathbf{V}_d |\Lambda|^{1/2} \right]_{i,:} \right\| = \sqrt{\sum_{j=1}^{d} [\mathbf{V}_d]_{i,j}^2 |\lambda_j|},$$

the weighted leverage score of a vertex i. Then, our elimination step is simple. Let $\hat{\mathbf{A}}_d$ be a subset of $\mathbf{A}_d$, where we have eliminated all vertices with $\ell_i \leq \frac{\eta}{3k\ell_1}$. Let $\mathbf{P}_{\mathcal{H}}$ be a diagonal matrix of 1s and 0s, with a 1 only in the (i,i) indices such that $\ell_i > \frac{\eta}{3k\ell_1}$. Then,

$$\hat{\mathbf{A}}_d = \mathbf{P}_{\mathcal{H}} \mathbf{A}_d \mathbf{P}_{\mathcal{H}}.$$

We can now guarantee the following upper bound on the error introduced by the elimination.

Proposition 12. *Let $\hat{\mathbf{A}}_d$ be created as above,*

$$\hat{\mathbf{A}}_d = \mathbf{P}_{\mathcal{H}} \mathbf{A}_d \mathbf{P}_{\mathcal{H}}.$$

where $\mathbf{P}_{\mathcal{H}}$ is a diagonal matrix of 1s and 0s, with a 1 on (i,i) indices such that $\ell_i > \frac{\eta}{3k\ell_1}$. Then,

$$\frac{\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}}{k} - \eta \leq \frac{\mathbf{1}_{\mathcal{X}}^T \hat{\mathbf{A}}_d \mathbf{1}_{\mathcal{Y}}}{k} \leq \frac{\mathbf{1}_{\mathcal{X}}^T \mathbf{A}_d \mathbf{1}_{\mathcal{Y}}}{k} + \eta, \tag{B.24}$$

for all subsets of k vertices $\mathcal{X}, \mathcal{Y}$.

Proof. Let for brevity θ be a user tuned threshold. Moreover, let $\mathbf{P}_{\mathcal{H}}$ be a diagonal matrix of 1s and 0s, with a 1 on (i,i) indices such that $\ell_i > \theta$, and let $\mathbf{P}_{\mathcal{L}}$ be a diagonal matrix of 1s and 0s, with a 1 on (i,i) indices such that $\ell_i \leq \theta$. Clearly,

$$\mathbf{P}_{\mathcal{H}} + \mathbf{P}_{\mathcal{L}} = \mathbf{I}_{n\times n}.$$

Then, we can rewrite $\mathbf{A}_d$ as:

$$\mathbf{A}_d = (\mathbf{P}_{\mathcal{H}} + \mathbf{P}_{\mathcal{L}})\mathbf{A}_d(\mathbf{P}_{\mathcal{H}} + \mathbf{P}_{\mathcal{L}}) = \mathbf{P}_{\mathcal{H}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}} + \mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}} + \mathbf{P}_{\mathcal{H}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}} + \mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}}.$$

Then, we have the following

$$\begin{aligned}|\mathbf{1}_{\mathcal{X}}^T(\mathbf{A}_d - \hat{\mathbf{A}}_d)\mathbf{1}_{\mathcal{Y}}| &= |\mathbf{1}_{\mathcal{X}}^T(\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}} + \mathbf{P}_{\mathcal{H}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}} + \mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}})\mathbf{1}_{\mathcal{Y}}| \\ &\leq |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}| + |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{H}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}| + |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}| \end{aligned} \tag{B.25}$$

Observe that for the first error term we have

$$\begin{aligned}|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}| &= |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\Lambda_d^{1/2}\mathbf{S}\Lambda_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}| \\ &\leq \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\Lambda_d^{1/2}\| \cdot \|\mathbf{S}\Lambda_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}\| \\ &\leq \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\Lambda_d^{1/2}\| \cdot \|\mathbf{S}\| \cdot \|\Lambda_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}\| \\ &= \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\Lambda_d^{1/2}\| \cdot 1 \cdot \|\Lambda_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{L}}\mathbf{1}_{\mathcal{Y}}\| \leq \sqrt{k}\cdot\theta\cdot\sqrt{k}\cdot\theta = k\cdot\theta^2. \end{aligned} \tag{B.26}$$

where the second and third inequalities come due to the Cauchy-Schwarz inequality. In the above $\mathbf{S}$ denotes the diagonal matrix that contains the signs of the eigenvalues. Clearly, its operator norm is 1. Hence, the last inequality in the above is due to the fact that $\mathbf{1}_{\mathcal{X}}, \mathbf{1}_{\mathcal{Y}}$ have k entries with 1, and each picks the rows of $\mathbf{V}_d\Lambda_d^{1/2}$ with the highest leverage score. Then, due to the triangle inequality on the k-largest row norms (i.e., leverage scores) of $\mathbf{V}_d\Lambda_d^{1/2}$ we get the final result.

Similarly, we can bound the remaining two error terms

$$\begin{aligned}|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{A}_d\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}| &= |\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\Lambda_d^{1/2}\mathbf{S}\Lambda_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}| \\ &\leq \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\Lambda_d^{1/2}\| \cdot \|\mathbf{S}\Lambda_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}\| \\ &\leq \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\Lambda_d^{1/2}\| \cdot \|\mathbf{S}\| \cdot \|\Lambda_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}\| \\ &= \|\mathbf{1}_{\mathcal{X}}^T\mathbf{P}_{\mathcal{L}}\mathbf{V}_d\Lambda_d^{1/2}\| \cdot 1 \cdot \|\Lambda_d^{1/2}\mathbf{V}_d^T\mathbf{P}_{\mathcal{H}}\mathbf{1}_{\mathcal{Y}}\| \\ &\leq \sqrt{k}\cdot\theta\cdot\sqrt{k}\cdot\ell_1 = k\cdot\theta\cdot\ell_1. \end{aligned} \tag{B.27}$$

Since, $\theta \leq \ell_1$, we conclude that the above error can be bounded as

$$\frac{|\mathbf{1}_{\mathcal{X}}^T(\mathbf{A}_d - \hat{\mathbf{A}}_d)\mathbf{1}_{\mathcal{Y}}|}{k} \leq 3 \cdot k \cdot \theta \cdot \ell_1 = \eta.$$

Hence, we obtain the proposition. □

B.8 NP-hardness of DkS on rank-1 matrices

In this section, we establish the hardness of the quadratic formulation of DkS, even for rank-1 matrices. Interestingly the problem is not hard when we relax it to its bilinear form as we showed in our main result. The claim follows.

Lemma 19. *DkS is NP-hard for rank-1 matrices* $\mathbf{A}$ *with one negative eigenvalue.*

Proof. Observe that a rank-1 matrix with 1 negative eigenvalue can be written as

$$\mathbf{A} = -\mathbf{v}\mathbf{v}$$

where $\lambda_1 = \|\mathbf{v}\|^2$. Then, see that

$$\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T\mathbf{A}\mathbf{1}_{\mathcal{S}} = \max_{|\mathcal{S}|=k} -\mathbf{1}_{\mathcal{S}}^T\mathbf{v}\mathbf{v}^T\mathbf{1}_{\mathcal{S}} = \min_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T\mathbf{v}\mathbf{v}^T\mathbf{1}_{\mathcal{S}} = \left(\min_{|\mathcal{S}|=k} |\mathbf{1}_{\mathcal{S}}^T\mathbf{v}|\right)^2 = \left(\min_{|\mathcal{S}|=k} \left|\sum_{i\in S} v_i\right|\right)^2.$$

An algorithm that can solve the above problem, can be used to solve SUBSETSUM. In SUBSETSUM we are given a set of integers and we wish to decide whether there exists a non-empty subset of these integers that sums to zero. In the following algorithm we show how this can be trivially done, by solving $\min_{|\mathcal{S}|=k} \left|\sum_{i\in S} v_i\right|$ for all values of k. If for some value of k the sum in the optimizaton is zero, then we decide YES as the output for the SUBSETSUM. Hence, solving $\max_{|\mathcal{S}|=k} \mathbf{1}_{\mathcal{S}}^T\mathbf{A}\mathbf{1}_{\mathcal{S}}$ is NP-hard even for rank-1 matrices, in the general case. □

B.9 Additional Experiments

In Fig. B.1, we show additional experiment on 9 more large-graphs. The description of the graphs can be found in Table B.1. Moreover, in Fig. B.2. The description of the experiments can be found in the figure captions.

Algorithm 13 SubsetSum via rank-1 DkS

Input: $\mathbf{v} = [v_1, \ldots, v_n]$
for $i = 1 : k$ **do**
  $s_i = \min_{|\mathcal{S}|=k} \left|\sum_{i \in S} v_i\right|$
  if $s_i == 0$ **then**
    Output: `YES`
  else
    Output: `NO`
  end if
end for

Table B.1: Datasets used in our experiments

DATA SET	NODES	EDGES	DESCRIPTION
COM-DBLP (SNAP.STANFORD.EDU)	317K	1M	DBLP COLLABORATION NETWORK
COM-LIVEJOURNAL (SNAP.STANFORD.EDU)	3.9M	34.6M	SOCIAL NETWORK
WEB-NOTREDAME (SNAP.STANFORD.EDU)	325K	1.4M	WEB GRAPH
EGO-FACEBOOK (SNAP.STANFORD.EDU)	4K	88K	SOCIAL NETWORK
CA-ASTROPH (SNAP.STANFORD.EDU)	18K	396K	COLLABORATION NETWORK
CA-HEPPH (SNAP.STANFORD.EDU)	12K	237K	COLLABORATION NETWORK
CA-CONDMAT (SNAP.STANFORD.EDU)	23K	186K	COLLABORATION NETWORK
CA-GRQC (SNAP.STANFORD.EDU)	5K	28K	COLLABORATION NETWORK
CA-HEPTH (SNAP.STANFORD.EDU)	9K	51K	COLLABORATION NETWORK
LOC-BRIGHTKITE (SNAP.STANFORD.EDU)	58K	214K	SOCIAL NETWORK
ROADNET-CA (SNAP.STANFORD.EDU)	1.9M	5.5M	ROAD NETWORK
EMAIL-ENRON (SNAP.STANFORD.EDU)	36K	367K	EMAIL COMMUNICATION NETWORK
COM-ORKUT (SNAP.STANFORD.EDU)	3M	117M	SOCIAL NETWORK
FLICKR (KONECT.UNI-KOBLENZ.DE)	105K	2.3M	COMMON METADATA NETWORK
FBMEDIUM (KONECT.UNI-KOBLENZ.DE)	63K	817K	SOCIAL NETWORK

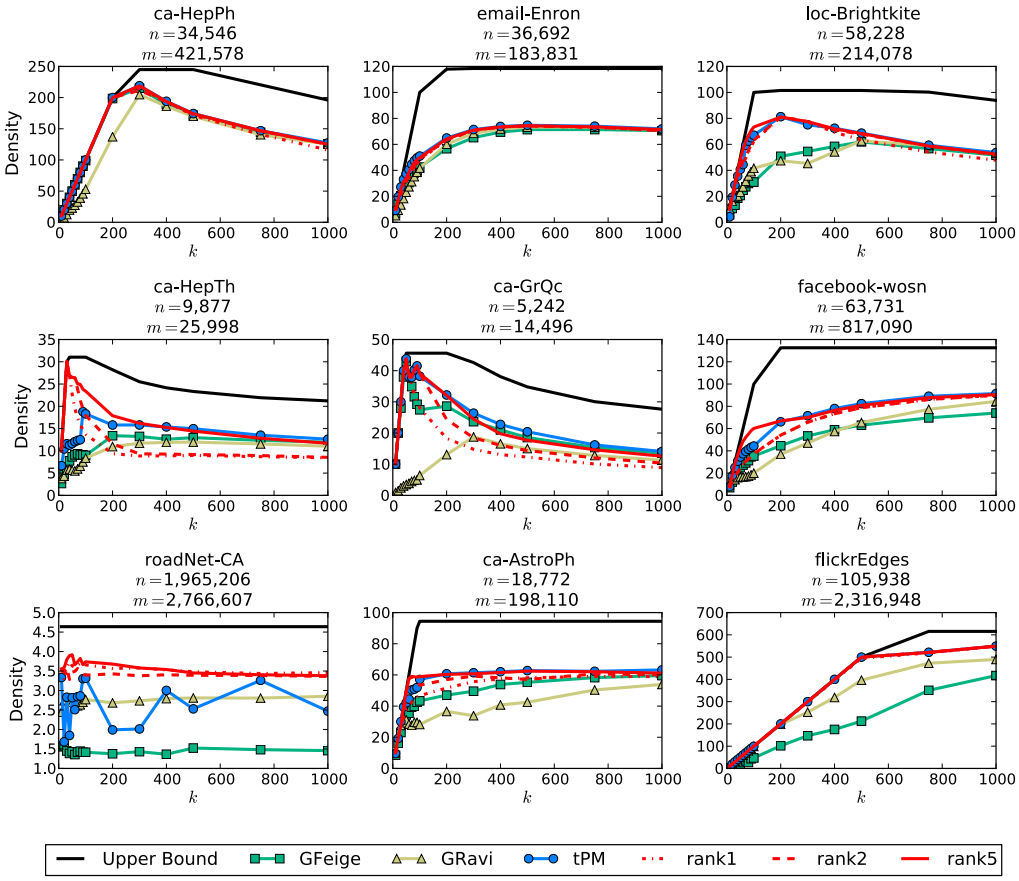


Figure B.1: Subgraph density vs. subgraph size (k). We show the comparison of densest subgraph algorithms on several additional datasets: Academic collaboration graphs from Arxiv (ca-HepTh, ca-HepPh, ca-GrQc, Ca-Astro), Geographic location-based networks (roadNet, loc-Brightkite), The Enron email communication graph (email-Enron) and a facebook subgraph (facebook-wosn). The number of vertices and edges are shown in each plot. As can be seen, in almost all cases rank-2 and rank-5 spannograms match or outperform previous algorithms. One notable exception is the ca-GrQc where, for subgraphs of size above $k = 400$ or above, T-power performs better. Another observation is that the spannogram benefits are often more significant for smaller subgraph sizes. It can also be seen that the tightness of our data-dependent bound (solid black line) varies for different data sets and subgraph sizes.

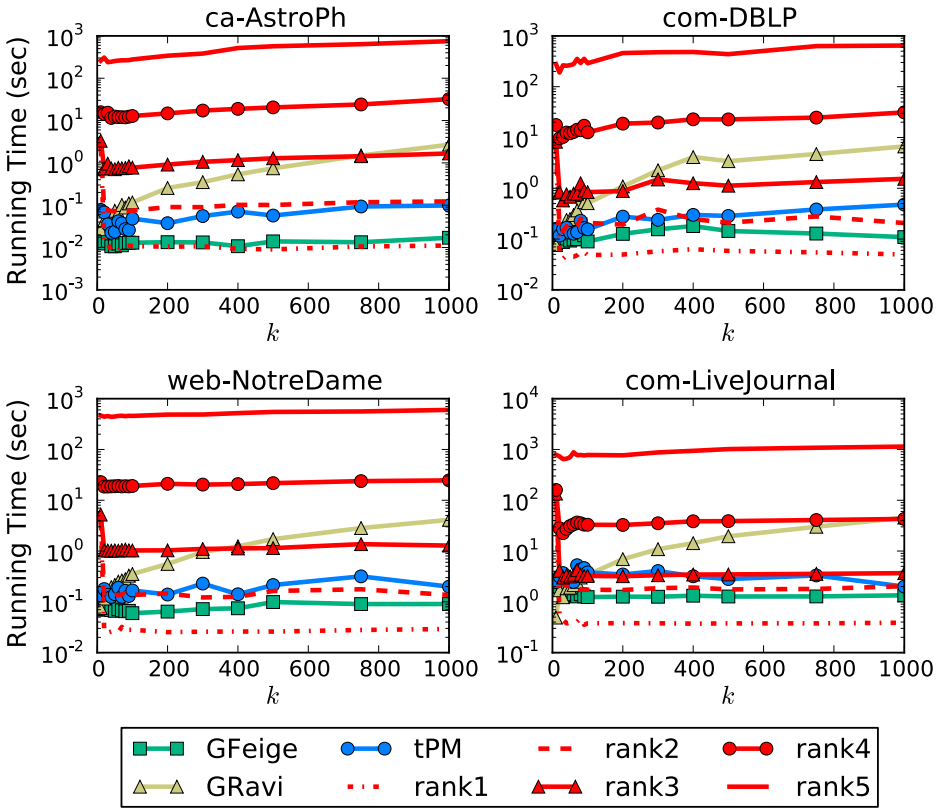


Figure B.2: Running times on a MacBook Pro 10.2, with Intel Core i5 @ 2.5 GHz (2 cores), 256 KB L2 Cache per core, 3 MB L3 Cache, and 8 GB RAM. Experiments were run on MATLAB R2011b (7.13.0.564). As can be seen, Rank-1 is significantly faster than all other algorithms for all tested cases. Rank-2 is comparable to prior work, having running times of a few seconds. Rank-5 was the highest accuracy setting we tested. It can take several minutes on large graphs and seems useful only when high accuracy is desired or other methods are far from the upper bound. The approximation error in the ϵ-net was set to 0.1.

Bibliography

[1] Amazon Web Services, Elastic Map Reduce. `http://aws.amazon.com/elasticmapreduce/`.

[2] The coding for distributed storage wiki. `http://tinyurl.com/storagecoding`.

[3] Mrjob. `http://pythonhosted.org/mrjob/`.

[4] Rudolf Ahlswede, Ning Cai, S-YR Li, and Raymond W Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204–1216, 2000.

[5] Noga Alon, Troy Lee, Adi Shraibman, and Santosh Vempala. The approximate rank of a matrix and its algorithmic applications: approximate rank. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, pages 675–684. ACM, 2013.

[6] Brendan PW Ames. *Convex relaxation for the planted clique, biclique, and clustering problems*. PhD thesis, University of Waterloo, 2011.

[7] A.A. Amini and M.J. Wainwright. High-dimensional analysis of semidefinite relaxations for sparse principal components. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 2454–2458. IEEE, 2008.

[8] Sanjeev Arora, David Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of np-hard problems. In *STOC*, 1995.

[9] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.

[10] M. Asteris, D.S. Papailiopoulos, and G.N. Karystinos. Sparse principal component of a rank-deficient matrix. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 673–677. IEEE, 2011.

[11] Bahman Bahmani, Ravi Kumar, and Sergei Vassilvitskii. Densest subgraph in streaming and mapreduce. *Proceedings of the VLDB Endowment*, 5(5):454–465, 2012.

[12] Quentin Berthet and Philippe Rigollet. Optimal detection of sparse principal components in high dimension. *arXiv preprint arXiv:1202.5070*, 2012.

[13] Quentin Berthet and Philippe Rigollet. Complexity theoretic lower bounds for sparse principal component detection, 2013.

[14] Quentin Berthet and Philippe Rigollet. Computational lower bounds for sparse pca. *arXiv preprint arXiv:1304.0828*, 2013.

[15] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. Detecting high log-densities: an $O(n^{1/4})$ approximation for densest k-subgraph. In *STOC*, 2010.

[16] Christos Boutsidis, Michael W Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 968–977. Society for Industrial and Applied Mathematics, 2009.

[17] V. R. Cadambe, C. Huang, S. A. Jafar, and J. Li. Optimal repair of mds codes in distributed storage via subspace interference alignment. *arXiv preprint arXiv:1106.1250*, 2011.

[18] Viveck R Cadambe, Cheng Huang, and Jin Li. Permutation code: optimal exact-repair of a single failed node in mds code based distributed storage systems. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 1225–1229. IEEE, 2011.

[19] Viveck R Cadambe, Cheng Huang, Jin Li, and Sanjeev Mehrotra. Polynomial length mds codes with optimal repair in distributed storage. In *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*, pages 1850–1854. IEEE, 2011.

[20] Viveck R Cadambe, Syed A Jafar, and Hamed Maleki. Distributed data storage with minimum storage regenerating codes-exact and functional repair are asymptotically equally efficient. *arXiv preprint arXiv:1004.4299*, 2010.

[21] Viveck R CADAMBE and Syed Ali JAFAR. Interference alignment and degrees of freedom of the k-user interference channel. *IEEE transactions on information theory*, 54(8):3425–3441, 2008.

[22] J. Cadima and I.T. Jolliffe. Loading and correlations in the interpretation of principle compenents. *Journal of Applied Statistics*, 22(2):203–214, 1995.

[23] T Tony Cai, Zongming Ma, and Yihong Wu. Sparse pca: Optimal rates and adaptive estimation. *arXiv preprint arXiv:1211.1309*, 2012.

[24] Tony Cai, Zongming Ma, and Yihong Wu. Optimal estimation and rank detection for sparse spiked covariance matrices. *arXiv preprint arXiv:1305.3235*, 2013.

[25] F. Chung, L. Lu, and V. Vu. Eigenvalues of random power law graphs. *Annals of Combinatorics*, 7(1):21–33, 2003.

[26] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2001.

[27] Daniel Cullina, Alexandros G Dimakis, and Tracey Ho. Searching for minimum storage regenerating codes. *arXiv preprint arXiv:0910.2245*, 2009.

[28] A. d'Aspremont, F. Bach, and L.E. Ghaoui. Optimal solutions for sparse principal component analysis. *The Journal of Machine Learning Research*, 9:1269–1294, 2008.

[29] A. d'Aspremont, F. Bach, and L.E. Ghaoui. Approximation bounds for sparse principal component analysis. *arXiv preprint arXiv:1205.0121*, 2012.

[30] A. d'Aspremont, L. El Ghaoui, M.I. Jordan, and G.R.G. Lanckriet. A direct formulation for sparse pca using semidefinite programming. *SIAM review*, 49(3):434–448, 2007.

[31] Alexandre d'Aspremont, Francis R. Bach, and Laurent El Ghaoui. Full regularization path for sparse principal component analysis. In *Proceedings of the 24th international conference on Machine learning*, ICML '07, pages 177–184, 2007.

[32] Alexandre d'Aspremont et al. Weak recovery conditions using graph partitioning bounds. 2010.

[33] I.S. Dhillon and D.S. Modha. Concept decompositions for large sparse text data using clustering. *Machine learning*, 42(1):143–175, 2001.

[34] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *Information Theory, IEEE Transactions on*, 56(9):4539–4551, 2010.

[35] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh. A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3):476–489, 2011.

[36] Johann Peter Gustav Lejeune Dirichlet. Beweis des satzes, dass jede unbegrenzte arithmetische progression, deren erstes glied und differenz ganze zahlen ohne gemeinschaftlichen factor sind, unendlich viele primzahlen enthält. In *Abhand. Ak. Wiss. Berlin 48*, volume 48, 1837.

[37] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense communities in the web. In *WWW*, 2007.

[38] Salim El Rouayheb and Kannan Ramchandran. Fractional repetition codes for repair in distributed storage systems. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1510–1517. IEEE, 2010.

[39] Uriel Feige and Michael Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *Journal of Algorithms*, 41: 174–211, 2001.

[40] Uriel Feige, David Peleg, and Guy Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[41] B. Gawalt, Y. Zhang, and L. El Ghaoui. Sparse pca for text corpus summarization and exploration. *NIPS 2010 Workshop on Low-Rank Matrix Approximation*, 2010.

[42] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *PVLDB*, 2005.

[43] Alex Gittens, Prabhanjan Kambadur, and Christos Boutsidis. Approximate spectral clustering via randomized sketching. *arXiv preprint arXiv:1311.2854*, 2013.

[44] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin. On the locality of codeword symbols. *Information Theory, IEEE Transactions on*, 58(11):6925–6934, 2011.

[45] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[46] J. Han and L. A. Lastras-Montano. Reliable memories with subline accesses. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 2531–2535. IEEE, 2007.

[47] Tracey Ho, Muriel Médard, Ralf Koetter, David R Karger, Michelle Effros, Jun Shi, and Ben Leong. A random linear network coding approach to multicast. *Information Theory, IEEE Transactions on*, 52(10):4413–4430, 2006.

[48] R.A. Horn and C.R. Johnson. *Matrix analysis*. Cambridge university press, 1990.

[49] Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(suppl 1):i213–i221, 2005.

[50] C. Huang, M. Chen, and J. Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*, pages 79–86. IEEE, 2007.

[51] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in windows azure storage. In *USENIX Annual Technical Conference (USENIX ATC)*, 2012.

[52] Cheng Huang. LRC erasure coding in windows storage spaces. In *Storage Developer Conference (SDC)*, 2013.

[53] Vinay Jethava, Anders Martinsson, Chiranjib Bhattacharyya, and Devdatt Dubhashi. The lovasz theta function, svms and finding large dense subgraphs. In *NIPS*, 2012.

[54] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2005.

[55] I.T. Jolliffe. Rotation of principal components: choice of normalization constraints. *Journal of Applied Statistics*, 22(1):29–35, 1995.

[56] I.T. Jolliffe, N.T. Trendafilov, and M. Uddin. A modified principal component technique based on the lasso. *Journal of Computational and Graphical Statistics*, 12(3):531–547, 2003.

[57] M. Journée, Y. Nesterov, P. Richtárik, and R. Sepulchre. Generalized power method for sparse principal component analysis. *The Journal of Machine Learning Research*, 11:517–553, 2010.

[58] H.F. Kaiser. The varimax criterion for analytic rotation in factor analysis. *Psychometrika*, 23(3):187–200, 1958.

[59] G. M. Kamath, N. Prakash, V. Lalitha, and P. V. Kumar. Codes with local regeneration. *arXiv preprint arXiv:1211.1932*, 2012.

[60] George N Karystinos and Athanasios P Liavas. Efficient computation of the binary vector that maximizes a rank-deficient quadratic form. *IEEE Trans. IT*, 56(7):3581–3593, 2010.

[61] O. Khan, R. Burns, J. Plank, and C. Huang. In search of i/o-optimal recovery from disk failures. In *Proceedings of the 3rd USENIX conference on Hot topics in storage and file systems*, pages 6–6. USENIX Association, 2011.

[62] Subhash Khot. Ruling out ptas for graph min-bisection, densest subgraph and bipartite clique. In *FOCS*, 2004.

[63] Donald E Knuth, Ron Graham, Oren Patashnik, et al. Concrete mathematics. 1989.

[64] Volodymyr Kuleshov. Fast algorithms for sparse principal component analysis based on rayleigh quotient iteration. 2013.

[65] Rudolf Lidl and Harald Niederreiter. Finite fields: Encyclopedia of mathematics and its applications. *Computers & Mathematics with Applications*, 33(7):136–136, 1997.

[66] Jimmy Lin and Michael Schatz. Design patterns for efficient graph algorithms in mapreduce. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*, pages 78–85. ACM, 2010.

[67] L. Mackey. Deflation methods for sparse pca. *Advances in neural information processing systems*, 21:1017–1024, 2009.

[68] Michael W Mahoney and Petros Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

[69] Xiangrui Meng and Michael W Mahoney. Robust regression on mapreduce. *ICML 2013, (to appear)*.

[70] M. Mihail and C. Papadimitriou. On the eigenvalue power law. *Randomization and approximation techniques in computer science*, pages 953–953, 2002.

[71] B Miller, N Bliss, and P Wolfe. Subgraph detection using eigenvector l1 norms. In *NIPS*, 2010.

[72] B. Moghaddam, Y. Weiss, and S. Avidan. Generalized spectral bounds for sparse lda. In *Proceedings of the 23rd international conference on Machine learning*, pages 641–648. ACM, 2006.

[73] B. Moghaddam, Y. Weiss, and S. Avidan. Spectral bounds for sparse pca: Exact and greedy algorithms. *Advances in neural information processing systems*, 18:915, 2006.

[74] B. Moghaddam, Y. Weiss, and S. Avidan. Fast pixel/part selection with sparse eigenvectors. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[75] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010.

[76] D. S. Papailiopoulos and A. G. Dimakis. Locally repairable codes. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 2771–2775. IEEE, 2012.

[77] D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe. Repair optimal erasure codes through hadamard designs. volume 59, pages 3021–3037, 2013.

[78] Dimitris S Papailiopoulos and Alexandros G Dimakis. Distributed storage codes through hadamard designs. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 1230–1234. IEEE, 2011.

[79] Dimitris S Papailiopoulos, Alexandros G Dimakis, and Stavros Korokythakis. Sparse pca through low-rank approximations. *arXiv preprint arXiv:1303.0551*, 2013.

[80] N. Prakash, G. M. Kamath, V. Lalitha, and P. V. Kumar. Optimal linear codes with a local-error-correction property. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 2776–2780. IEEE, 2012.

[81] K. V. Rashmi, N. B. Shah, and P. V. Kumar. Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction. *Information Theory, IEEE Transactions on*, 57(8):5227–5239, 2011.

[82] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran. Explicit construction of optimal exact regenerating codes for distributed storage. In *Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on*, pages 1243–1249. IEEE, 2009.

[83] KV Rashmi, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster. In *USENIX HotStorage*, 2013.

[84] Sekharipuram S Ravi, Daniel J Rosenkrantz, and Giri K Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

[85] A.S. Rawat, O.O. Koyluoglu, N. Silberstein, and S. Vishwanath. Optimal locally repairable and secure codes for distributed storage systems. *arXiv preprint arXiv:1210.6954*, 2012.

[86] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1100–1124, 2009.

[87] Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Research in Computational Molecular Biology*, pages 456–472. Springer, 2010.

[88] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing elephants: Novel erasure codes for big data. *Proceedings of the VLDB Endowment*, 2013.

[89] Nihar B Shah, KV Rashmi, P Vijay Kumar, and Kannan Ramchandran. Interference alignment in regenerating codes for distributed storage: Necessity and code constructions. *Information Theory, IEEE Transactions on*, 58(4):2134–2158, 2012.

[90] Nihar B Shah, KV Rashmi, P Vijay Kumar, and Kannan Ramchandran. Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff. *Information Theory, IEEE Transactions on*, 58(3):1837–1852, 2012.

[91] H. Shen and J.Z. Huang. Sparse principal component analysis via regularized low rank matrix approximation. *Journal of multivariate analysis*, 99(6):1015–1034, 2008.

[92] Kenneth W Shum and Yuchong Hu. Exact minimum-repair-bandwidth cooperative regenerating codes for distributed storage systems. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 1442–1446. IEEE, 2011.

[93] B.K. Sriperumbudur, D.A. Torres, and G.R.G. Lanckriet. Sparse eigen methods by dc programming. In *Proceedings of the 24th international conference on Machine learning*, pages 831–838. ACM, 2007.

[94] Anand Srivastav and Katja Wolf. *Finding dense subgraphs with semidefinite programming*. Springer, 1998.

[95] C. Suh and K. Ramchandran. Exact-repair mds code construction using interference alignment. *Information Theory, IEEE Transactions on*, 57(3):1425–1442, 2011.

[96] Changho Suh and Kannan Ramchandran. On the existence of optimal exact-repair mds codes for distributed storage. *arXiv preprint arXiv:1004.4663*, 2010.

[97] Akiko Suzuki and Takeshi Tokuyama. Dense subgraph problems with output-density conditions. In *Algorithms and Computation*, pages 266–276. Springer, 2005.

[98] James Joseph Sylvester. Thoughts on inverse orthogonal matrices, simultaneous signsuccessions, and tessellated pavements in two or more colours, with applications to newton's rule, ornamental tile-work, and the theory of numbers. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 34(232):461–475, 1867.

[99] I. Tamo, Z. Wang, and J. Bruck. Mds array codes with optimal rebuilding. In *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pages 1240–1244. IEEE, 2011.

[100] Itzhak Tamo, Dimitris S Papailiopoulos, and Alexandros G Dimakis. Optimal locally repairable codes and connections to matroid theory. *arXiv preprint arXiv:1301.7693*, 2013.

[101] Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. *arXiv preprint arXiv:1011.3027*, 2010.

[102] Zhiying Wang, Itzhak Tamo, and Jehoshua Bruck. On codes for optimal rebuilding access. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 1374–1381. IEEE, 2011.

[103] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. *Peer-to-Peer Systems*, pages 328–337, 2002.

[104] Yunnan Wu and Alexandros G Dimakis. Reducing repair traffic for erasure coding-based storage via interference alignment. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 2276–2280. IEEE, 2009.

[105] Aaron D Wyner. Random packings and coverings of the unit n-sphere. *Bell System Technical Journal*, 46(9):2111–2118, 1967.

[106] Raymond W Yeung, Shuo-Yen Robert Li, Ning Cai, and Zhen Zhang. Network coding theory: single sources. *Communications and Information Theory*, 2(4):241–329, 2005.

[107] Xiao-Tong Yuan and Tong Zhang. Truncated power method for sparse eigenvalue problems. *arXiv preprint arXiv:1112.2679*, 2011.

[108] Y. Zhang, A. d'Aspremont, and L.E. Ghaoui. Sparse pca: Convex relaxations, algorithms and applications. *Handbook on Semidefinite, Conic and Polynomial Optimization*, pages 915–940, 2012.

[109] Y. Zhang and L. El Ghaoui. Large-scale sparse principal component analysis with application to text data. *Advances in Neural Information Processing Systems*, 2011.

[110] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.

Vita

Dimitris Papailiopoulos received the Diploma and M.Sc. degrees in electronic and computer engineering from the Technical University of Crete, Greece, in 2007 and 2009, respectively. Between 2009 and 2012, he was a graduate student with the Department of Electrical Engineering, University of Southern California. Since January 2013, he has been with the Department of Electrical and Computer Engineering at the University of Texas at Austin, where he is currently working towards the Ph.D. degree. His research interests are in the areas of coding theory, information theory, and large-scale data processing. In particular, codes for distributed storage and spectral techniques for large-scale graph analytics.

Permanent address: 1044 Camino La Costa
Austin, Texas 78758

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.